

EE209 Final Report: Replicating Hand Writing Gestures via Leap Motion Controlled Robotic System

Ryan Hamidi
Alia Nasrallah
Jiahui Li
Xuerui Yan

December 8, 2016

Abstract

The work of this paper presents the development of a controlled robotic system that can accurately track hand motion used in writing and accomplish gesture replication. This is implemented with the utilization of Leap Motion, an Arduino Uno microcontroller, JavaScript software implementation, a user hand motion simulation interface, and robotic actuator systems: including an OWI robotic arm and XY Plotter. Process control and communication between different system components is aided through the incorporation of node.js runtime environment, and the leap.js and johnny-five.js libraries. Algorithm optimization has been conducted in order to demonstrate the following key experimentation results: 1) inverse kinematics makes gesture replication challenging for a kinematic chain robotic arm operating with DC motors, 2) using multiple hand features detected by Leap Motion allows for more accurate hand writing motion tracking and gesture replication, 3) using linear approximation algorithms for drawing curves allows for better tracking of more complex hand writing motions, 4) using a trajectory tracker greatly aids a user's experience with our writing system, and 5) storing data about the robotic actuator systems' response to motion detection facilitates the completion of gesture replication.

1 Introduction

1.1 Problem Statement

It is challenging to mass reproduce text or illustrations that involve unique human dexterity. The question then becomes how can we utilize robotics and/or software to easily and accurately recreate hand gestures that identify personal handwriting. Utilizing motion capture technology and market available products, we wish to create an algorithm that will track hand motion and gestures to replicate the unique handwriting of an individual.

1.2 Motivation and Scope

Gesture control is a recent and evolving technology, where developers need large capacity synthesis and interpretation of user requirements [1]. Therefore, an opportunity arises which leads to the need for experimenting with new methods of interaction that primarily exists in virtual reality and science fiction.

Robotics provide an efficient approach in the development of assistant devices because of their high precision and functionality, as well as their ease of control and maneuverability. Therefore, mechanical, electrical and software building blocks can be used to build articulated mechanical systems and control gesture recognition systems. We will utilize a physical actuator system, 3D interaction, and gesture recognition sensors to create a solution for our problem statement.

Our project aims to reduce the cost and time spent by artists to replicate their original work and individuals who must sign many documents. As a longer term goal, developing this system could allow recreation of personal handwriting and drawing on more atypical materials such as canvas, wood,

cloth, glass. Additionally, our system aims to implement low cost technology to facilitate more advanced, practical, and resourceful ways to interact with your computer. We also wish to create a system that replicates more intricate movements as opposed to other robotic arms that are more coarse in motion.

To develop our project, we will be focusing on the following areas:

Areas we will investigate:

- Integrating Leap Motion to control a mechanical system and communicate with a microcontroller.
- Retaining extracted data from the Leap Motion controller.
- Replicating human handwriting to a whiteboard.
- Using available consumer products that are low cost and comprehensible amongst the general customer market.

We will leave the following areas for future development:

Areas we will not investigate:

- Inputs such as EMG, voice control, keyboard commands, or text that is already programmed into our robot.
- Our end effector being able to grab or exchange its writing tool.
- Retaining multiple people's handwriting styles at the same time or storage of writing motion data.

2 Software and Hardware Implementation

2.1 Motion Capture Sensor

Leap Motion Controller The Leap Motion controller provided the means to recognize and control hand gestures. The detection of different hand gestures is accomplished with the Leap Motion device, in which the movement of the human hand and fingers were observed in 3D, and the x, y, and z positions and velocities were recorded. The leap motion controller device and its associated coordinate system is shown in Figure 1.

The Leap Motion controller employs a right-handed Cartesian coordinate system. The origin is centered at the top of the Leap Motion Controller, where positive x is to the right, positive y is upwards, and positive z is directed towards the user.



Figure 1: Leap Motion Controller Device [2] and Coordinate System [3], where positive x is to the right, positive y is upwards, and positive z is directed towards the user

The sensor is capable of processing 40 frames per second to create a 3D dot pattern of the detected hand with its three infrared cameras and two monochrome cameras. These cameras observe a field of view of 150° over 0.25 mm^3 interactive 3D space structured as a hemisphere centered on the device. The effective range of the Leap Motion controller extends from approximately 25 mm to 600 mm above the device. The power supply and data transfer (into software that can further analyze hand information) is accomplished via the USB port.

Units of Leap Motion In order to communicate between Leap Motion controller and our robotic systems, we need to know the data acquired from Leap Motion. Every frame acquired from Leap Motion is a set of data which is written in JSON format. Leap Motion measures these physical quantities with the following units.

Quantities	Units
Distance:	Millimeters
Speed	Millimeters/Second
Time	Microsecond (unless otherwise noted)
Angle	Radians

Table 1: The Units Used By Leap Motion

JSON stands for JavaScript Object Notation; which is a lightweight data-interchange format. It is comprehensible by humans to read and write and is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming language that is familiar to programmers of the C-family of languages. These properties make JSON an ideal data-interchange language.[\[4\]](#)

Why was Leap Motion Selected For our system, the leap motion sensor was utilized as opposed to other market-available gesture recognition devices because it is easy to handle and relatively inexpensive. Although the leap motion sensor operates over a small area, it was still able to provide a suitable range for hand movements, and thus was suitable for our purposes.

When integrated with Arduino Uno, the x, y, and z positions and velocities of the hands and fingers can be extracted from leap motion sensor data to control the OWI robotic arm or the XY plotter motors, and thus achieve the exact same motion observed by the sensor. This leads to the replication of the hand movement through the controlled robotic system.

Leap Motion Limitations When testing the leap motion controller, we came across several issues: first, it was difficult for us to write in 3D space, and so a glass surface was needed to mimic a writing surface. However, we soon discovered that leap motion's performance and accuracy degraded when going through the clear glass surface.

Additionally, we were required to conduct several trials and extensive testing to get the appropriate thresholds needed for our robotic system movements. The leap motion sensor also suffered from sporadic instability in detecting movements along a specific axis, which resulted in inaccuracy when replicating the intended motion. Furthermore, since the leap motion controller measures coordinates linearly, rotational joints were difficult for us to control, requiring us to come up with optimization techniques to replicate curvatures in writing.

2.2 Arduino Uno Micro-controller

To develop our system, the Arduino Uno board was selected as our microcontroller. The reason we selected Arduino Uno is because there were available online resources for us to make use of, since it is open source hardware and free software. Despite its simplicity, it is versatile because it has multiple PWM and I/O pins, I2C communication buses, and multiple ways of implementing coding algorithms.

We have the option of using either Arduino IDE or by connection via a separate script language to implement our algorithms and control the motion capture device. The serial communication between different platforms and our gesture recognition subsystem can be developed using this microcontroller. Furthermore, it is compatible with many different hardware components, including motor shields. The Arduino Uno work environment makes it it a relatively simple embedded controller board for us to integrate in our system.

The Arduino Uno board technical specifications is listed in table 2. A picture of the board is shown in figure 2.



Figure 2: Arduino Uno Board [5]

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Table 2: Arduino Uno Board Specifications [6]

2.3 Software implementation

Interaction with the Leap motion device can be performed through the integration of the Arduino Uno board and our JavaScript software model. For this report, our software model is defined as the implementation of our script code in JavaScript that controls our systems. Thus managing the data acquired by the motion capture device. To get data from the leap motion device and send it to the Arduino Uno board, the node.js, leap.js and johnny-five libraries were used.

2.3.1 Node.js

Node.js is a JavaScript runtime environment provided by NPM (the package manager for JavaScript). This construct uses an event-driven, non-blocking I/O model that makes it efficient by allowing multiple hardware connections to be handled concurrently [7].

2.3.2 Leap.js

The leap.js library enables the integration between JavaScript and the leap motion controller. Leap.js works by connecting to a websocket server running locally through the leap motion sensor service [3]. About every 10 ms the service sends a frame, consisting of JSON data including the position of hands and fingers, over the websocket.

JSON is built on two structures: (1) a collection of name/value pairs such as object, record, struct, and dictionary and (2) an ordered list of values such as an array, vector, list, or sequence. In JSON, the data may take the form of object, array, value, string, or number. [4]. JSON evaluates a response to an event and returns a JavaScript object. The JSON data is parsed in a strict manner and any malformed JSON is rejected and a parse error is thrown. [8] A `dataType` is the type of data you expect back from the server, and the Leap Motion controller's `dataType` is a JSON object. This means after a frame is processed, it should always at least return an empty JSON object or null from any request in order for jQuery to accept it as valid. [9]

Leap Motion Objects The controller object in leap.js mediates the connection between the application and the leap motion sensor. The controller can be created using the `var controllerLM = new Leap.Controller()` command (as shown in Figure 3).

The controller waits for a leap 'frame' to be transmitted from the motion capture device, which then calls a function that is used to set our variables for later use with johnny-five. The `Frame` class represents a set of hand and finger tracking data detected in a single frame. A single frame is the amount of time in which Leap Motion acquires information such as position, orientation, and motion from the hand for synthesis. The Leap Motion sensor reports these frames at the Leap frame rate. [10] A frame can be thought of as the root of our leap motion data model.

Leap Motion APIs The leap motion API can measure physical quantities, such as distance in mm, time in microseconds, speed in mm/s, and angle in radians [3].

Within the Leap Motion API, `Hand` is a class that provides information about the physical characteristics of a detected hand, the arm to which the hand is attached, and lists of the fingers associated with the hand [3]. Hand tracking information includes data on palm position and velocity, vectors for the palm normal and direction to the fingers, and many other features.[11]

Fingers are represented by the `Fingers` class to provide information about each finger on a hand. Fingers can be identified by thumb, index, middle, ring and pinky [3]. The `Pointable` class also reports the physical characteristics of the detected fingers, but also physical tools, such as a pen [3].

The data model extracted from the leap frames can contain several classes, as defined above. These classes include attributes [3] that were considered in our project's development.

- The hand palm position attribute (`hand.palmPosition`) is a 3 element array representing a unit direction vector. The center position of the palm in mm corresponds to the leap motion controller origin.
- The hand palm velocity attribute (`hand.palmVelocity`) is a 3 element array representing a vector. It describes the rate of change of the palm position in mm/s.
- The hand palm width attribute (`hand.palmWidth`) is a number representing the average outer width of the hand (not including the thumb and fingers) in mm.
- The hand palm normal attribute (`hand.palmNormal`) is a 3 element array representing a unit direction vector, which corresponds to the normal vector to the palm. If the hand is flat facing the leap motion device, this vector will point downwards.

- The `hand.pointables[]` attribute is an array of pointable objects or fingers detected in a particular frame, which is associated with the hand.
- The pointable tip position attribute (`pointable.tipPosition`) is a 3 element array representing a position vector. The tip position in mm corresponds to the leap motion controller origin.
- The pointable tip velocity attribute (`pointable.tipVelocity`) is a 3 element array representing a vector. It describes the rate of change of the tip position in mm/s.

2.3.3 Johnny-five

Johnny-five is an open source JavaScript robotics framework [12]. The reason we selected Johnny-five as our platform is that it's capable of communicating with our Arduino Uno board and it provided a wide variety of libraries and functions for controlling hardware components. This makes it a more versatile framework than Cylon; which is what was initially used as our robotics framework because of quickly accessible online resources. Even though both are Node.js solutions, Johnny-five had the additional benefit of easily including all hardware components within an instantiation. The instantiation of the Arduino board and the leap motion controller are shown in Figure 3.

```
var Leap = require('leapjs');
var five = require('johnny-five');
var board = new five.Board();

var controllerLM = new Leap.Controller();
```

Figure 3: Instantiation of johnny-five and leap motion controller

3 Robotic Arm Implementations

3.1 Servo Motor Arm Challenges

The first robotic arm that was utilized in our project's development was a servo motor robotic arm, as shown in Figure 4. It was made of a wooden construction that sat on a wooden base and had a gripping end effector. It was lightweight and small, making it portable.

However, the arm only contained 3 DOF (i.e. 3 joints with a single degree of freedom). It could not perform rotational movement of its end effector and was deemed to be impractical for writing replication. This is because a human's wrist is a main contributor to an individual's handwriting. Challenges regarding its lack of maneuverability and functionality dissuaded the group to pursue further experimentation with this robotic arm.

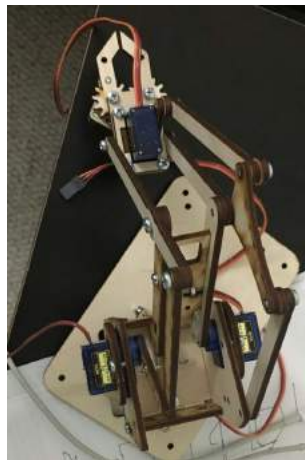


Figure 4: Servo motor Robotic Arm

3.2 OWI Robotic Arm Implementation

3.2.1 Background

Introduction to OWI Robotic Arm In order to conduct preliminary testing of the functionality of a robotic arm along with its compatibility with Leap Motion and Arduino, a robotic arm needed to be attained quickly. Without it, only abstract estimations could be done and this would not provide adequate initial analysis of our implementation. Our group decided to purchase a robotic arm as opposed to building one with raw materials. Although our implementation would be mandated by the manufacturer, the group felt that it would allow for a more sturdy tool and significantly reduce assembling time, allowing concentrated efforts elsewhere. The OWI Robotic Arm Edge was purchased based on these criteria conditions.

- Easy to assemble
- Inexpensive
- Robust enough to withstand multiple testing trials
- Resources available to aid in communication and functionality with Arduino microcontrollers

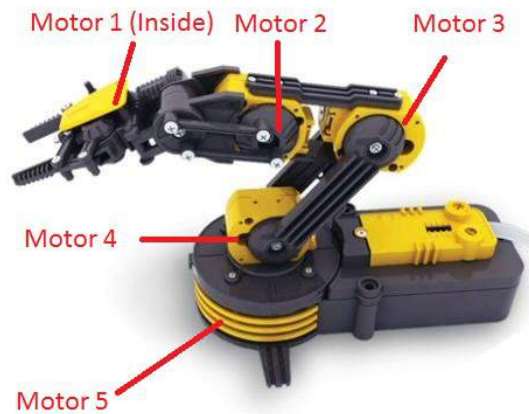


Figure 5: OWI Robotic Arm [13]

OWI Robotic Arm Operation

Construction The OWI Robotic Arm is shown in Figure 5. It is an inexpensive arm that has 4 DOF (i.e. each joint that have one angle of rotation). Analogous to the human arm, these include joints for wrist, elbow, shoulder translation and rotation. The arm could also be assembled quickly by following detailed instructions, and its structure removed the risk of fracture during testing.

Controller The OWI Robotic Arm is packaged with a controller to orient its position. As shown in Figure 6 it allows directional control of each joint. As labeled on the controller, these corresponding motors control these joints.

- Motor 1: Gripper
- Motor 2: Wrist
- Motor 3: Elbow
- Motor 4: Shoulder
- Motor 5: Base Rotation



Figure 6: OWI Robotic Arm Controller

The controller was optimal for allowing physical examination of its range of motion, limitations, as well as an optimal method for resetting the arm position for a new test trial.

DC Motors The OWI Robotic Arm contains 4 DC motors (as shown in Figure 7), which allow for rotational movement of each of the joints. The arm itself operates with 4 D-cell batteries for a total operating voltage of 6V. This meant that the robot arm was versatile to operate with different forms of power, including an external power supply, batteries, or an Arduino micro-controller. This proved to be useful as it allowed for versatile operation and did not require any inconvenient power supplies as some other robotic arms required.



Figure 7: The motor for the OWI Robotic Arm [14]

A standard DC motor needs two current supplies, one that supplies the magnetic field and the other that interacts with the magnetic field to generate the rotational motion. These are applied to the stator windings and the rotor windings, respectively. They utilize a commutator to feed current into the rotor windings. The speed is controlled by varying the rotor voltage and current, or by varying the magnetic flux that varies the current in the field windings. The OWI arm utilizes permanent magnet motors.[15]

Adafruit Motor Shield Due to the fact that there are 4 DC motors that control the operation of the OWI robotic arm, the Arduino Uno Microcontroller by itself is not capable of controlling all the motors. This is why a motor shield board had to be used with our Arduino microcontroller. The Adafruit V1 Motor Shield was used and is shown in 8. The reasoning for using this motor shield is described in the OWI Robotic Arm Methods section.

The main components of the Adafruit Motor Shield are two L293 Dual H-Bridge IC chips, 1 74HC595N Serial to Parallel output Latch, and its 6 analog inputs. The analog inputs are connected to A0-A5 analog pins of the Arduino Uno. [17]

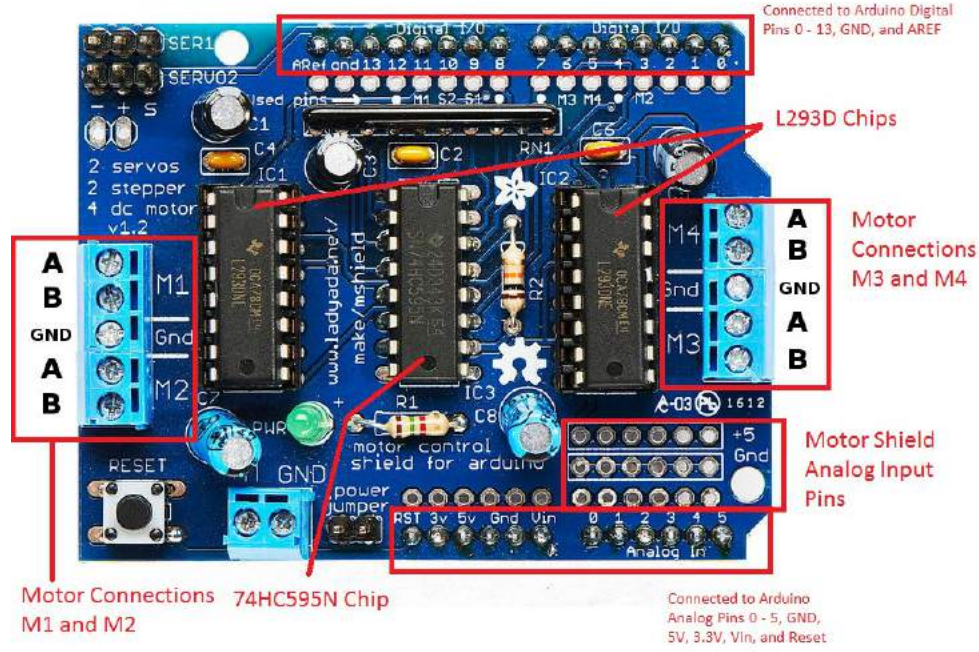


Figure 8: The Adafruit Motorshield Version 1 [16]

L293 Dual H-Bridge Each L293 chip can control up to two DC motors bi-directionally as long as they consume less than 600 mA of current. They are used in this motor shield because they contain built in kick-back diodes so that any perturbations in inductive impedance will not damage the driver, or any other components connected to it. It also contains a PWM input per driver to control the motor speed. It is operational with a 5V supply (same as the Arduino Uno supply), and optimal for motors that operate from 4.5V to 36V. [18]

A general H-bridge circuit is shown in Figure 9. It contains 4 switching elements (2 high side and 2 low side) and a load whose configuration resembles the english letter 'H'. The load in Figure 9 represents a motor which has resistive, inductive, and capacitive components (as shown in Figure 10). The switching elements are typically FET devices and the diodes are called catch diodes. The purpose of a catch diode is to synchronize timing of the on and off state of both the high and low side switches. It also facilitates better heat dissipation of the switches. [19]

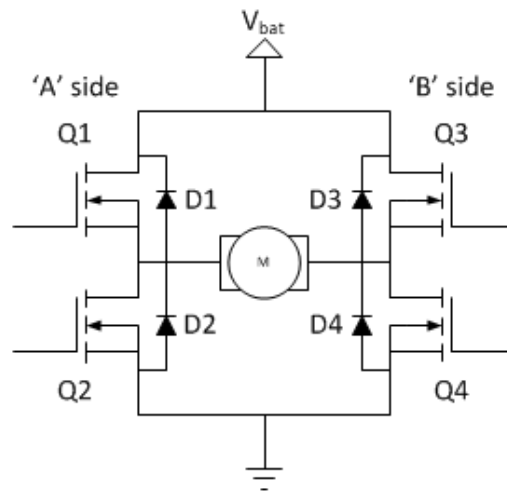


Figure 9: A General H-Bridge Circuit [19]

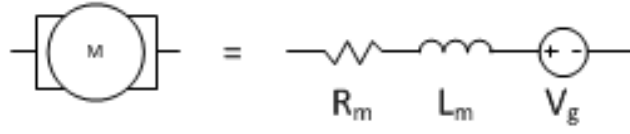


Figure 10: Motor Model of 8 [19]

Each chip contains four half H-bridges. A half H-bridge is one half of the general H-bridge circuit. It contains only one high side and one low side FET switch connected in a cascode configuration while still containing the catch diodes. The two switches turn on and off complementary to each other with a non-overlapping dead time. This means that there is no shoot-through condition that is prevalent in general H-bridge circuits. Shoot-through refers to a state of the driving circuit where a low-resistance path between power and ground exists. [20]

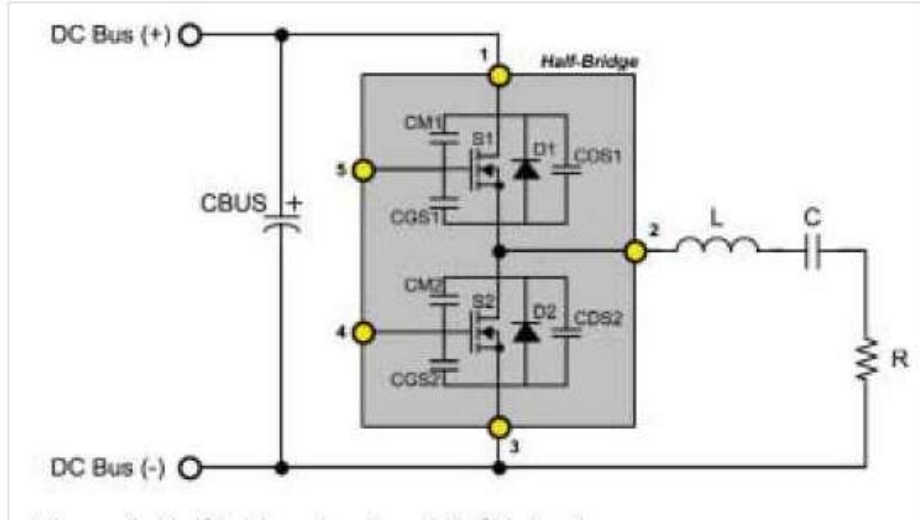


Figure 11: Half H-Bridge Circuit [20]

The L293D functional block diagram is shown in Figure 12 with the 4 half H-bridge circuits labeled as the triangular amplifier symbols. The truth table to "Side A" of the H-Bridge circuit is shown in Figure 13; with the truth table of "Side B" being identical. The logic diagram for the L293 chip is shown in Figure 14. The drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2 EN and drivers 3 and 4 enabled by 3,4 EN. When the enable is high, the associated drivers are enabled and their outputs are active in-phase with the inputs. When the enable is low, the drivers are disabled and their outputs are in the high-impedance off state. [21]

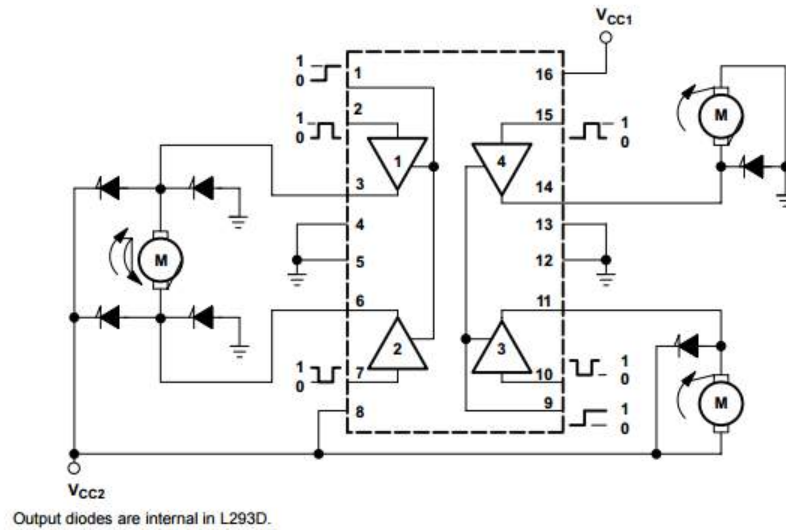


Figure 12: The L293D Functional Block Diagram [21]

Table 1. Function Table (Each Driver)⁽¹⁾

INPUTS ⁽²⁾		OUTPUT (Y)
A	EN	
H	H	H
L	H	L
X	L	Z

(1) H = high level, L = low level, X = irrelevant, Z = high impedance (off)

(2) In the thermal shutdown mode, the output is in the high-impedance state, regardless of the input levels.

Figure 13: The L293D Truth Table [21]

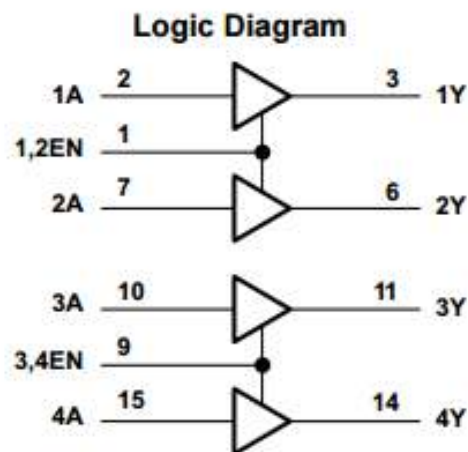


Figure 14: The L293D Logic Diagram [21]

74HC595N Serial to Parallel Output Latch The 74HC595N is an 8-bit serial-in/serial or parallel-out shift register with a storage register and 3-state outputs (high, low, or high-impedance off).

The device features a serial input and output to enable cascading and an asynchronous reset \overline{MR} input. \overline{MR} set to LOW will reset the shift register. Data is shifted on the LOW to HIGH transitions of the SHCP input and is transferred to the storage register on the LOW to HIGH transition of the STCP input. The data stored appears at the output when the output enable input \overline{OE} is LOW. Logic level HIGH on \overline{OE} causes the outputs to be set to a high-impedance OFF state. The functional diagram is shown in Figure 15, it's logic diagram is shown in Figure 16, and it's pin layout is shown in Figure 17. [22]

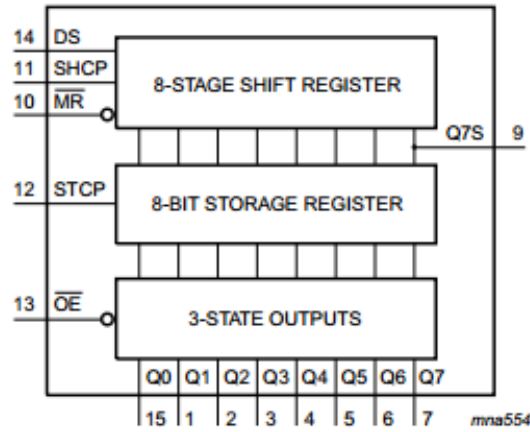


Figure 15: 74HC595N Functional Diagram [22]

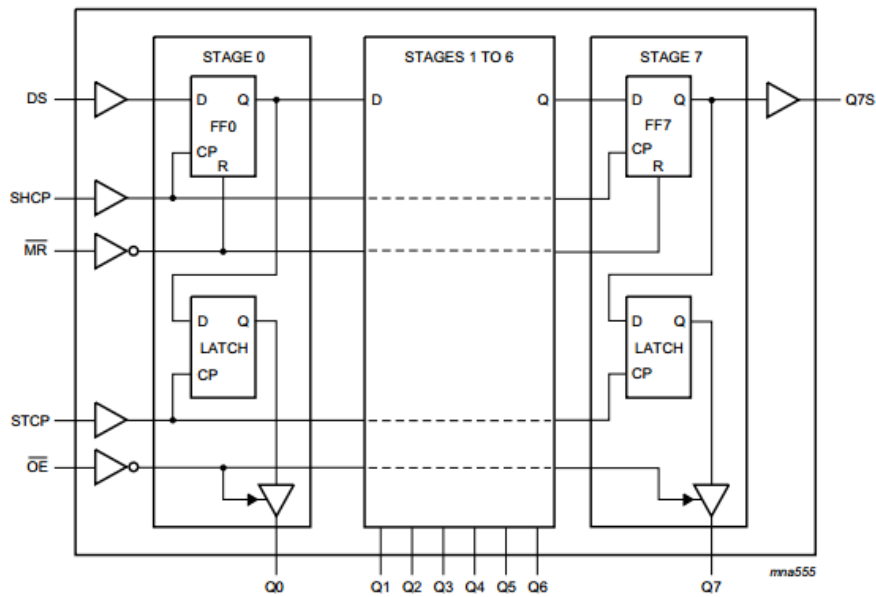


Figure 16: 74HC595N Logic Diagram [22]

The Adafruit Motor Shield's compatibility with the Arduino Uno microcontroller is achievable through proper utilization of the 74HC595N's pin assignments. On the Arduino Uno, there are pins designated as dataPin (pin 14), clockPin (pin 11), and latchPin (pin 8). These pins are connected to the Arduino microcontroller at pins 11, 12, and 8 respectively. [23]

When paired with the Arduino Uno, the 74HC595N allows for the control of 8 outputs at a time through

the utilization of only 3 pins. It also allows for possible cascading of multiple shift registers. Synchronous serial communication allows for pulsing one pin up and down. Thereby communicating a data byte to the register bit by bit. The pulsing of the clock pin allows for delineation between bits. Once the entire byte is stored in the register, the HIGH or LOW data is output to an appropriate individual output pin; this is where the parallel output of its name originates from. The serial output portion of the component is facilitated from the extra pin that passes serial information through the microcontroller not modified. This means that if 16 bits (2 bytes) are transferred, the first byte will pass through the first register into the second register. [23]

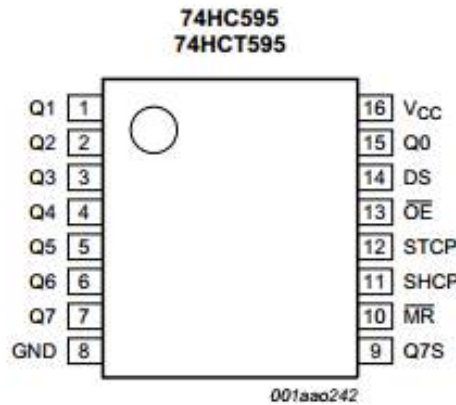


Figure 17: 74HC595N Pin Diagram [22]

Analog Inputs for Sensors The Arduino Uno has 6 analog input pins, labeled A0-A5. Each provides 10 bits of resolution. By default, they measure from ground to 5V, which is how 2 of the pin connections are used. The third pin connection is used as the analog input for each pin. This is the pin that is utilized for functions similar to Arduino IDE’s analogRead. Basically, read the value from a specified analog pin. [24]

3.2.2 Methods

OWI Arm Implementation with Arduino The Adafruit V1 Motor Shield is shown in Figure 8. This motor shield allows for easy platform to connect motors, servos, and stepper motors in robotics or mechatronics applications. For our particular implementation, it matches our 4 motor quantity requirement, is compatible with Arduino Uno, contains speed and direction controls, references are available on its operation with Arduino Uno, and is readily available by many distributors. Many manufacturers produce duplicate products of this motor shield, and the K1D Motor Shield was chosen due to its immediate availability. Functionally, it is identical to the Adafruit Motor Shield. [17]

Standard Firmata Firmata is a protocol for communicating with microcontrollers from software on a host computer. The protocol has the potential to be implemented in firmware on any microcontroller configuration as well as any computer software package. Currently the most complete implementation is for Arduino; which further supports why Arduino was the the preferred microcontroller. [25]

Firmata is based on midi message format in which command bytes are 8 bits and data bytes are 7 bits. The number of bytes in a message must conform with the corresponding midi message. The Johnny-Five node module is a client library that implements the Firmata protocol to communicate commands from a computer. [25]

There are two main methods of Firmata usage. One model uses various methods within the Arduino sketch to selectively send and receive data between the Arduino device and the software running on the

host computer. The second more common usage is to load a general purpose sketch called StandardFirmata on the Arduino board and then use the host computer exclusively to interact with the Arduino board. The standard DC motors that are used in the OWI robotic arm can be controlled using StandardFirmata and that is what is loaded onto the Arduino board. All of the Arduino operations are controlled in the JavaScript coding. For other components, it may be necessary to load different variants of the Firmata sketch. [26]

Connection with Arduino The Adafruit Motor Shield layout allows for easy connection with the Arduino Uno micro-controller. It is connected on top of the Arduino Uno and utilizes digital pins 0 to 13, AREF and GND (Figure 19) while utilizing analog pins 0 to 6, RESET, 3.3V, 5V, GND, and RESET (Figure 18).



Figure 18: Analog Pin Connections of Motor Shield to Arduino Uno



Figure 19: Digital Pin Connections of Motor Shield to Arduino Uno

Arm Implementation With Johnny-five

Motor Class The motor class constructs objects that represent a single motor. The class works with directional and non-directional motors and allows connection from external motor controllers or shields. This makes it optimal for utilization with the Adafruit Motor Shield. For standard motor declaration within this class, the necessary parameter to define is pin assignment (the address for the motor pin). Additionally, because the motor shield utilizes a shift register, additional register properties need to be defined. These include data, clock, and latch, as well as the bits (the switch bits to be flipped to control an Hbridge).[27]

Within our script, individual motors can be declared as shown in Figure 20. Each individual motor has its unique designation of PWM pins and bits. The shield survey for a L293D driven motor shield, which encompasses the Adafruit Version 1 motor shield and any of its clones, is shown in Figure 21. Since all 4 motor shield connections will be connected to the OWI arm's 4 motors, all pins listed will be utilized. [27]


```

var five = require("johnny-five"),
    board = new five.Board();

board.on("ready", function() {

    var motor = new five.Motor({
        pins: { pwm: 11 },
        register: { data: 8, clock: 4, latch: 12 },
        bits: { a: 2, b: 3 }
    });
});

```

Figure 20: Motor Class Declaration in Johnny-Five [27]

ShiftRegister													
Name	Register	Shield Config	Operating Voltage(1)	Max A	Stackable(2)								
SainSmart L293D Motor Drive Shield (clone of AdaFruit Motor Shield v1)	data: 8 clock: 4 latch: 12	ADAFRUIT_V1: {M1, M2, M3, M4}	4.5-10V	1.2A per motor	No								
	<table><tr><th>M1</th><th>M2</th><th>M3</th><th>M4</th></tr><tr><td>pins: { pwm: 11 } bits: { a: 2, b: 3 }</td><td>pins: { pwm: 3 } bits: { a: 1, b: 4 }</td><td>pins: { pwm: 6 } bits: { a: 5, b: 7 }</td><td>pins: { pwm: 5 } bits: { a: 0, b: 6 }</td></tr></table>					M1	M2	M3	M4	pins: { pwm: 11 } bits: { a: 2, b: 3 }	pins: { pwm: 3 } bits: { a: 1, b: 4 }	pins: { pwm: 6 } bits: { a: 5, b: 7 }	pins: { pwm: 5 } bits: { a: 0, b: 6 }
M1	M2	M3	M4										
pins: { pwm: 11 } bits: { a: 2, b: 3 }	pins: { pwm: 3 } bits: { a: 1, b: 4 }	pins: { pwm: 6 } bits: { a: 5, b: 7 }	pins: { pwm: 5 } bits: { a: 0, b: 6 }										

Figure 21: Motor Shield Survey of Adafruit Version 1 Motor Shield (And Any Of Its Clones) [27]

Sensor Class The sensor class constructs an object that represents a single analog sensor component attached to a physical board. The class is generic to allow for connection of many types of sensors. The parameters to be declared are pin, frequency, and threshold; which are described in Figure 22. An example of the declaration of a sensor is shown in Figure 23. Within this class its API contains two event handlers: change and data. The change event is emitted whenever the value of the sensor changes more than the threshold value allows. The data event is fired as frequently as the user defined freq (in milliseconds). Since our implementation will not set a threshold value, and the data event replaces the read event (from other API), our event handler for the sensors will be data. [28]

Property	Type	Value/Description	Default	Required
pin	Number, String	Analog Pin. The Number or String address of the pin the sensor is attached to, ie. "A0" or "I1"		yes
freq	Number	Milliseconds. The frequency in ms of data events.	25ms	no
threshold	Number	Any. The change threshold (+/- value).	1	no

Figure 22: Parameter Declarations of Sensor Class in Johnny-Five [28]


```
var temp = new five.Sensor({
  pin: "A0",
  freq: 250,
  threshold: 5
});
```

Figure 23: Example of Sensor Declaration in Johnny-Five [28]

Arm Control Algorithm In the previous sections, the physical arm, hardware utilization, components, software implementation, and libraries that allow proper operation have all been described. In this section, the algorithm for controlling the robotic arm based on all these descriptions will be analyzed. This is done by separating the arm control algorithm into two components; the motor control and sampling control.

Motor Control The first process in being able to capture and replicate hand motion used in writing with the robotic arm was establishing proper motor control. This is a crucial part of this implementation because without it, the arm's movements will be inaccurate and unpredictable. Multiple preliminary experiments had to be done to establish a foundation of how the motors will be controlled as well as developing fluency with JavaScript syntax in Johnny-Five for declaring hardware, components, and processing.

For our robotic arm, the motors had to respond to a single or combination of unique hand characteristic(s). Initially, the prescribed method of detecting a hand feature was `palmPosition`. However, when this was implemented, certain problems became apparent and challenges were not resolved.

Firstly, the Leap Motion sensor will always detect a position for the palm, even when the palm is not moving. This means that if the motors are triggered by `palmPosition`, even slight trembling and perturbations of a stationary hand will cause the motor to move. Secondly, there is not much uniqueness between the three values in the `palmPosition` array. This means that it would be difficult to distinguish which motors should move given hand movement. Thirdly, the position of the palm does not account for the speed of hand motion. Lastly, only using position makes it difficult to distinguish which direction the motor should rotate. Simply assigning rotation on positive and negative values is too restrictive on what movements can be accomplished. Among these challenges, controlling the immediate stoppage of the robotic arm and direction control was the most pressing issue to resolve.

These challenges prompted the change for the motion control condition to be controlled with `palmVelocity`. The `palmVelocity` has 3 velocities, one along each axis; i.e. x axis velocity, y axis velocity, and z axis velocity. This means that a unique definition of the hand in a specific direction can be defined by a specific array element to control a motor. It also allows for the stoppage of a motor to be stabilized because a stoppage of the hand will result in an almost zero velocity. Finally, the sign magnitude of `palmVelocity` can be used to uniquely define motor rotation direction because it detects which direction the hand is moving. [11]

Sampling Control The frame rate of the Leap Motion controller is approximately 40 FPS [30] while it is capable of storing 60 frames in its controller history. [31]. When optimizing this algorithm, it was discovered that frames were being synthesized by Leap Motion too frequently. This meant that the sensor was detecting the hand motion too frequently and over sampling. In situations when our hand was slow moving, the leap motion sensor would still detect its velocity and output it as a frame. When the motion of the motors was triggered on every processed frame, even a slow moving hand would cause motor rotation.

This is why a threshold velocity was set (in mm/sec). Different velocities were experimented with, and eventually a velocity of 10 mm/sec was analyzed to be optimal. This is because this speed closely resembles a moderate speed in which someone would write, could capture some intricate motions involved in writing, but also was not too slow as to create excessive frame processing. This is explained more elaborately in the testing/results section.

Even with accounting for this velocity modification, if motor rotation was triggered on every processed frame, it would change too frequently, i.e. even minor changes in hand velocity would rotate motors. This created staggered and unsteady movement. Generally, when people write, they maintain a constant speed. Taking this into consideration, the Leap Motion controller does not need to base its motor movement on each individual frame because the velocities within a certain interval of frames would be similar. This influenced the establishment of a count variable. The count would basically control the rate at which the motors would react to hand movement. This would allow us to limit how many frames the motors would respond to and create more stable movements in between batch frames being processed. This corrected a major problem that we were experiencing in which the motors would try to move forward and backward at the same time.

```

82
83
84 //moving motor 1
85 if (handvelocity1 > 10) {
86     //moving forward, no reverse count
87     count_motor1_reverse = 0;
88     //setting the count condition before motor
89     // moves
90     if (count_motor1_forward == 30){
91         console.log('motor 1 go forward');
92         motor1.forward(255);
93         //reset the count so that a new batch of
94         //frames can be processed
95
96         count_motor1_forward = 0;
97     } else {
98         //during the time that the next 30 are
99         //processed, we want to keep the same state of
100         //the motor
101         console.log('motor 1 keep moving forward');
102     }
103 } else if (handvelocity1 < -10){
104     //moving reverse, no forward count
105     count_motor1_forward = 0;
106     //setting the count condition before motor moves
107     if (count_motor1_reverse == 30){
108         console.log('motor 1 go reverse');
109         motor1.reverse(255);
110         //reset the count so that a new batch of frames
111         //can be processed
112         count_motor1_reverse = 0;
113     } else {
114         //during the time that the next 30 are processed,
115         //we want to keep the same state of the motor
116         console.log('motor 1 keep moving reverse');
117     }
118 } else {
119     //when our hands stop, we want to reset the value
120     //of count and make the motor stop
121     console.log('motor 1 stop');
122     motor1.stop();
123     count_motor1_forward = 0;
124     count_motor1_reverse = 0;
125
126 }
127

```

Figure 24: Motor Control in Arm Control Algorithm

Two counts are set for each motor, one for sampling forward rotation and one for sampling reverse rotation; resulting in a total of 8 counts in this algorithm. The way this algorithm operates is that once a hand velocity greater than the specified threshold of 10 mm/s is detected, based on whether it is positive or negative, it designates a direction for the motor (positive magnitude being forward and negative

magnitude being reverse direction). It then sets the opposite count to 0. The appropriate motor count will accumulate to 30, meaning on every frame the count increases by 1 and one sample batch will be set after 30 counts. Then the motor will move in the appropriate direction. It then sets that count back to zero so that a new set of frames can be accumulated. During this time that the next 30 frames are processed, the motor will maintain its rotation. When motion is slower than 10 mm/s, the motor will stop and both counts for that motor will reset to 0.

In the motor class API for Johnny-five, instantiating the rotation of the motor is accomplished with the code on lines 92 and 109 in Figure 24. Motor1 refers to a specific name of one of the 4 motors, forward or reverse specifies its direction, and the following value determines its speed. The speed value can be set to any value between 0 to 255. This value controls the voltage that is applied to the two pins of the motor. A value of 0 would be equivalent to motor.stop() command, while a speed of 255 would set a potential difference of Vcc on the DC motor. The higher the value, the higher the potential difference between the motors pins, and the faster the motor will move. [28] In initial testing and understanding of motor control, an isolated DC motor would rotate when this speed was set as low as 50. However, when implemented to the robotic arm, the extra weight from the robotic arm structure along with its power requirements mandated that this speed be set to the maximum value of 255.

If no hands are detected by the leap motion sensor, all counts are reset to zero. Upon termination of this algorithm, the motor declarations are re-established so that the motors can be commanded to stop simultaneously (even if a hand is still detected). This alleviated the challenge of preventing motors from running even after the code is terminated.

Workaround Hack Upon testing of this motor control algorithm, a particular error that kept re-occurring is shown in Figure 25. Due to an underlying issue regarding Leap Motion development and its communication with different servers, the Leap client and leapjs node module do not communicate well or sometimes is even unsafe when used with Node.js. When it randomly misses data, these frames are expected to be processed through the leapjs library, but it cannot determine the data within these frames because it is empty. This development error results in code crashing, abrupt termination, and component malfunction.

A workaround hack is an understood programming slang for a quick fix for an issue that may be occurring within running script, but does nothing to fix the underlying issue. As a hack, the error occurrences can be caught inside the handleData method. as shown in Figure 26. It is instantiated after the inclusion of the node module libraries and the board and controller declaration beginning on line 8. This workaround hack does not prevent the errors from occurring, but it prevents the code from terminating and malfunctioning. This allowed us to conduct testing and analysis without the interruption of randomly missed data.

```
SyntaxError: Unexpected end of JSON input
    at Object.parse (native)
    at BaseConnection.handleData (C:\Users\ryanh\node_modules\leapjs\lib\connection\base.js:135:22)
    at WebSocket.<anonymous> (C:\Users\ryanh\node_modules\leapjs\lib\connection\node.js:19:49)
    at emitTwo (events.js:106:13)
    at WebSocket.emit (events.js:191:7)
    at Receiver.self._receiver.oncontext (C:\Users\ryanh\node_modules\leapjs\node_modules\ws\lib\WebSocket.js:544:10)
    at Receiver.finish (C:\Users\ryanh\node_modules\leapjs\node_modules\ws\lib\Receiver.js:397:14)
    at Receiver.expectHandler (C:\Users\ryanh\node_modules\leapjs\node_modules\ws\lib\Receiver.js:384:31)
    at Receiver.add (C:\Users\ryanh\node_modules\leapjs\node_modules\ws\lib\Receiver.js:93:24)
    at Socket.firstHandler (C:\Users\ryanh\node_modules\leapjs\node_modules\ws\lib\WebSocket.js:524:22)
```

Figure 25: JSON Error

```

1
2  var Leap = require('leapjs');
3  var five = require('johnny-five');
4  var board = new five.Board();
5
6  var controllerLM = new Leap.Controller();
7
8  controllerLM.on('connect', function() {
9    var origHandleData = this.connection.handleData;
10   this.connection.handleData = function(data) {
11     try {
12       return origHandleData.call(this, data);
13     } catch (e) {
14       console.log(e);
15     }
16   };
17 });
18

```

Figure 26: JSON Workaround Hack for Missing Data [29]

3.3 OWI Robotic Arm Testing

3.3.1 Sampling Control Testing

For the arm control algorithm, the necessity of a threshold velocity and frame count parameters for sampling control were discussed. In this section, the testing procedures are described. Through analysis, results that verify the parameter values will describe how this aided the arm control algorithm of the OWI robotic arm.

Sampling Control Methods

Velocity Threshold Testing This test was conducted in order to determine what velocity rate should be set as a condition for Leap Motion to detect a moving hand. Coincidentally, this threshold calibration should include most of our collected data from testing trials, meaning the Leap Motion Controller can capture the various motions used in hand writing. Without it, even a slow moving hand would trigger frame processing. This results in over-processing of hand information, possibly utilizing stationary hand information as a motor rotation condition, and inaccurate motion replication.

The testing was designed such that velocities on each axis would be examined based on hand movements along each independent axis in both directions. The movement conditions are the following: (1) x axis movement includes leftward and rightward motion, (2) y axis movement includes upward and downward motion, and (3) the z axis includes forward and backward rotation. All axes and directions were considered because we did not want our system to be subjective to a particular axis movement, while encompassing different motions utilized for writing, and identifying if particular axes were more accurate than others.

The group's original postulation was that when the hand moves along a certain velocity, that specific axis velocity (called the focus axis velocity) would be much larger than the other two axis velocities; as our focus would not be movement along the other two axes. Therefore we would only see slight variations. While conducting testing, the intended direction of movement was logged to view what these variations would be. This is viewable by the reader in the red highlighted cells under the direction header of Table 3 in the Sampling Control Results section.

Frame Count Testing This will designate what optimal frame count would reduce the over-processing of hand information, better control the reaction rate of motors to hand movements, and appropriately capture different hand writing motions.

The testing was conducted such that motor rotation was triggered after waiting for a certain number of frames of a detected hand moving along one axis. We repeat this test for slow and fast moving hands, but not having a specific target velocity in mind (so long as it was above the verified velocity threshold

of 10 mm/s). This would give us variety of sample inputs and make our system more robust to various detected hand motion inputs.

If the user moves their hand in the opposite direction of each axis, it does not greatly affect the system's operation. It would only change the direction of motor rotation and the magnitude of the distance and velocity. This was confirmed through preliminary testing of interfacing Johnny-five with DC motors. Testing along one direction of each axis reduces the number of test trials and facilitates quicker testing. Therefore, the absolute value of the velocity and distance were logged.

For setting a count value, our hypothesis was the higher the count value, the longer the Leap Motion controller would take to output a processed frame. This is because it has to spend more time accumulating frames. For the user, this means extending the range or decreasing the speed they move their hand. However the user's maximum hand motion range is defined by the controller's hemispherical area for hand detection. Also, if the hand velocity is too slow, adequate accumulation is not possible, it may not be reflective of people writing, and may not exceed the velocity threshold.

The value of the frame count can be set as low as 0. However, setting the count to 0 would be the same as the Leap Motion Controller's default frame processing. Slightly increasing this value (to a value less than or equal to 5) would reduce the processing rate slightly, but it would not significantly improve our intention of reducing frame processing.

Sampling Control Results

Velocity Threshold Testing Upon examining the velocity data for each axis under the blue header in Table 3, the variations along all axes velocities cannot be considered slight. For each trial, data sets can be found in which certain velocities other than the focus axis velocity are the largest. As an example, for rows 1 to 12, the intended axis movement was the x axis, but in rows 4 and 5 the y axis and z axis respectively were greater than the x axis velocity. Similar situations can be found for the other two axes. This infers to the group that the hand velocity varies along all axes.

This was the first indication of the possible inaccuracies Leap Motion may have in detecting hand movements. This velocity inaccuracy is common along velocities that were slower. For velocities in which the focus axis velocity was higher than 100 mm/s, the focus axis velocity was larger than the other velocities. This led the group to believe that when the hand moves slower, a human will incorporate more calibration to maintain a slow consistent movement. This requires concentration and when the tester wants to adjust their hand position, will do so quickly. Unintentionally, this sudden movement gives a sudden increase in the velocity. The perturbations of re-adjusting the hand prevent the different axis velocities to be small. This implies that the Leap Motion controller is more accurate when the hand moves quickly.

Another factor to consider is that when the hand moves away from the Leap Motion sensor, usually the sensor will have a high velocity reading. The group postulates that the reason is as the hand pulls away from the sensor, it has less hand feature information to acquire. The amount of information decreases quickly and as it becomes less available the sensor gives erroneous high readings.

Given all of this information, the intention of this test was not to ensure that the focus axis velocity would be the largest velocity. Therefore solutions to velocity inaccuracies were not addressed. Also, because this was the first major form of testing conducted, it was not understood how large these inaccuracies would affect the overall system. For the interest of proceeding with testing, progressing implementation, and to observe how these observations would affect our robotic arm's movement, a threshold velocity needed to be defined from Table 3.

Trial	Time (ms)	Direction			Distance (mm)			Velocity (mm/s)		
		x	y	z	x	y	z	x	y	z
1	335	LEFT	UP	BACKWARD	170.31	57.524	105.46	1023.13	574.036	68.187
2	5180	RIGHT	DOWN	BACKWARD	240.24	26.56	19.2	25.68	18.19	2.75
3	9136	RIGHT	DOWN	FORWARD	354.81	59.58	46.89	19.65	4.57	24.6
4	8214	RIGHT	DOWN	FORWARD	357.53	1.45	41.7	23.18	37.31	18.74
5	15.56	RIGHT	UP	FORWARD	372.61	10.69	13.04	27.38	11.41	105.21
6	23.43	RIGHT	DOWN	BACKWARD	384.03	10.02	56.82	39.58	26.33	36.66
7	16.26	LEFT	DOWN	BACKWARD	283.23	39.15	12.07	25.85	52.98	39.56
8	22.38	LEFT	DOWN	FORWARD	380.82	25.81	10.9	17.96	75.14	29.5
9	21.99	LEFT	DOWN	BACKWARD	318.85	27.66	72.79	30.92	22.29	24.61
10	6.27	LEFT	DOWN	BACKWARD	348.15	54.86	109.94	8.15	5.93	26.65
11	7.39	LEFT	UP	BACKWARD	302.46	1.16	91.43	43.74	83.51	234.33
12	9.54	LEFT	DOWN	BACKWARD	418.17	14.27	90.5	154.98	104.82	55.62
13	29.69	RIGHT	UP	FORWARD	14.3	414.47	70.54	22.3	44.45	32.68
15	25.4	LEFT	UP	FORWARD	133.83	341.4	61.53	11.01	8.4	3.79
16	28.25	RIGHT	UP	FORWARD	24.58	494.19	117.79	10.83	4.16	4.17
17	6.93	LEFT	UP	FORWARD	33.47	472.9	100.38	17.7	170.52	5.01
18	8.97	LEFT	UP	FORWARD	18.73	474.77	65.21	4.82	64.29	1.64
19	7.36	LEFT	UP	FORWARD	141.3	340.99	174.82	351.42	414.77	141.53
20	7.85	RIGHT	DOWN	FORWARD	5.53	230.49	13.58	8.99	45.96	36
21	7.47	RIGHT	DOWN	FORWARD	22.56	406.85	6.07	49.1	85.52	1.85
22	18.04	LEFT	UP	FORWARD	40.81	7.01	146.49	47.86	96.67	17.8
23	6.85	RIGHT	UP	FORWARD	147.15	11.68	18.92	7.55	4.97	10.39
24	4.84	RIGHT	UP	FORWARD	109	24.86	91.67	44.13	84.79	104.07
25	37.01	RIGHT	UP	BACKWARD	128.3	20.71	178.56	69.46	22.07	27.8
26	8.44	RIGHT	UP	BACKWARD	208.85	117.18	348.88	14.04	117.33	73.26

Table 3: Table of Velocity Threshold Testing Data

After observing the data shown in Table 4 if the threshold velocity was set to 10 mm/s, then the Leap Motion Controller captures 80% of the data set. This is because most of the hand motions did not have velocities that registered below 10 mm/s. Even though a velocity threshold of 5 mm/s captures more of the testing data, the motors still move too frequently and motor control is difficult. We see a large decrease in data capture between a threshold of 10 mm/s and 15mm/s, which decreases further with higher velocities. Once the threshold is set to 30 mm/s, the controller captures less than 50% of the data. This implied to the group that most movements involved in hand writing are above 10 mm/s. This is an adequate threshold as it is capable of capturing enough writing motion gestures, prevents excessive motor rotation, and does not exclude too much data from our testing trials.

Velocity (mms/s)	Amount of Hand Motions This Threshold Can Capture (%)
5	88.00
10	80.00
15	73.33
20	65.33
25	57.33
30	48.00

Table 4: Amount of Velocity Threshold Testing Data That is Above Various Velocities

Frame Count	Direction	Trial	Time Lapse	Duration (ms)	Velocity (mm/s)	Total Counts	Total Time (ms)	Average Time Duration (ms)	Average Velocity (mm/s)
10	X-Right	1	1	80	25.04	60	512	86.4	61.69
			2	86	92.91				
			3	87	47.82				
			4	93	53.98				
			5	80	45.8				
			6	86	42.9				
10	X-Right	2	1	214	57.88	60	786	114.4	105.142
			2	148	353.37				
			3	164	20.33				
			4	80	27.52				
			5	85	36.03				
			6	95	30.58				
10	Z-FORWARD	1	1	7774	16.41	60	830	166	23.772
			2	93	20.32				
			3	286	13.29				
			4	111	12.15				
			5	95	29				
			6	245	27.69				
10	Z-FORWARD	2	1	82	160.31	60	572	98	183.186
			2	91	184.63				
			3	95	130.27				
			4	96	145.63				
			5	92	157.53				
			6	116	137.56				
10	Y-UP	1	1	8836	10.43	60	1120	224	15.124
			2	298	14.64				
			3	109	14.16				
			4	90	11.11				
			5	253	14.57				
			6	370	10.71				
10	Y-UP	2	1	3225	127.08	60	452	90.4	177.194
			2	90	125.36				
			3	91	116.8				
			4	90	157.93				
			5	91	167.13				
			6	90	191.67				

Table 5: Frame Count = 10 Testing Data

Frame Count Testing Table 5 shows the test data for a frame count equal of 10. For each direction, the group acquired 5 frame intervals from two trials. This is because the group felt this is a reasonable sample size to acquire data from and see variations. This is also why the Time Lapse column has 6 entries. It takes time to place our hand over the controller and for it to detect a hand, so the hand data capture start time was not always at the initialization of the testing code. We were only concerned with intervals and time duration and so starting time does not matter. From each frame interval, a time duration and velocity were obtained. For 5 frame intervals, summations and averages for the total time, time duration, and velocity were calculated.

For each frame interval, the values for total time, average time duration, and average velocity that Leap Motion detected have quite large deviations. Different hand velocities and directions alter the frame interval time duration and total processing time of 5 frame intervals. Especially for Y-axis variations, these results prove that a frame count of 10 would be insufficient in producing accurate motor reaction to hand movements and reducing Leap Motion processing.

The next frame count value to be tested for was 30. This can be seen in Table 6. The method of conducting this testing as well as the testing environment was the same as for when the frame count was set to 10. Even though the average time duration for each frame interval is longer, this is to be expected. This is because more frames need to be accumulated before a new frame is outputted by the controller.

Frame Count	Direction	Trial	Time Lapse	Duration (ms)	Velocity (mm/s)	Total Counts	Total Time (ms)	Average Time Duration (ms)	Average Velocity (mm/s)
30	X-Right	1	1	602	17.3	180	1910	261.6	28.006
			2	260	18.87				
			3	260	23.87				
			4	269	23.11				
			5	260	28.78				
			6	259	28.1				
30	X-Right	2	1	567	114.292	180	1902	267	101.8208
			2	278	110.262				
			3	260	83.28				
			4	260	86.26				
			5	260	64.9				
			6	277	50.11				
30	Z-FORWARD	1	1	351	84.42	180	1727	275.2	98.792
			2	272	110.62				
			3	271	106.54				
			4	290	51.86				
			5	271	71.42				
			6	272	69.1				
30	Z-FORWARD	2	1	1364	114.98	180	2759	279	100.612
			2	290	119.73				
			3	271	75.94				
			4	272	54.11				
			5	289	72.22				
			6	273	66.08				
30	Y-UP	1	1	1186	135.52	180	2575	277.8	113.814
			2	288	104.99				
			3	271	78.41				
			4	271	67.19				
			5	271	88.05				
			6	288	94.91				
30	Y-UP	2	1	3145	15.17	180	4562	283.4	19.192
			2	271	17.46				
			3	271	16.11				
			4	353	11.31				
			5	235	22.21				
			6	287	13.7				

Table 6: Frame Count = 30 Testing Data

Upon analysis of the data set, the controller's acquisition of data and frame processing had a much higher level of accuracy. This is because of consistent time duration detection. For different velocities across different axes, the average time duration remained approximately the same. This span ranges about 21.8 ms compared to the span of 137.6 ms in the frame count = 10 testing. Also, the average total time it took to determine 5 frame intervals was about 2.5 seconds. We felt this was an adequate time for our system to respond to a succession of hand movements, reduce motor reaction rate to hand movements, and facilitate more efficient processing. All of this would accomplish our goal of reducing the over-processing of unnecessary hand information.

While these results facilitate better processing, at least one other frame count needed to be tested for. This would determine if the trend of increasing the frame rate would produce even more consistent results. Our hypothesis was that there would eventually be an optimal count that produces the best results.

The test data for a frame count of 50 is shown in Table 7. Consistency that was evident with a frame count of 30 is not present, indicating deviation from our performance goal. For different velocities, the average time duration of each frame interval ranges a span of more than 6 seconds. The total time to process five frame intervals greatly increased; in one case exceeding 30 seconds. While conducting testing, it was difficult to move the hand to account for a long processing parameter. Especially for the y axis, the hand needed to move slowly and it was challenging to allow enough frame counts to accumulate. One reason for this is because since negative y would imply positions underneath the Leap Motion controller (outside of its viewing range), we effectively had half of the area and less time to move along the y axis. These levels of inconsistency and long time duration indicates that a frame count of 50 is less efficient for our system than a frame count of 30.

Frame Count	Direction	Trial	Time Lapse	Duration (ms)	Velocity (mm/s)	Total Counts	Total Time (ms)	Average Time Duration (ms)	Average Velocity (mm/s)
50	X-Right	1	1	1282	18.8	300	3858	515.2	18.0112
			2	434	15.14				
			3	824	12.686				
			4	433	13.12				
			5	451	15.75				
			6	434	14.56				
50	X-Right	2	1	1690	51.07	300	3893	440.6	50.07
			2	451	47.27				
			3	434	36.4				
			4	433	45.46				
			5	451	43.7				
			6	434	26.45				
50	Z-BACKWARD	1	1	4128	37.85	300	6884	551.2	28.604
			2	904	29.72				
			3	470	17.03				
			4	453	24.49				
			5	460	18.53				
			6	469	15.4				
50	Z-BACKWARD	2	1	7493	122.46	300	9787	458.8	124.312
			2	470	105.03				
			3	451	81.65				
			4	452	68.72				
			5	470	115.28				
			6	451	128.42				
50	Y-DOWN	1	1	11341	35.41	300	32294	6458.8	49.654
			2	7887	30.95				
			3	6757	34.33				
			4	6407	57.09				
			5	6651	49.55				
			6	4592	40.94				
50	Y-DOWN	2	1	7554	27.14	300	26562	5312.4	49.28
			2	4592	34.32				
			3	6029	52.31				
			4	5408	37.61				
			5	5940	36.85				
			6	4593	58.17				

Table 7: Frame Count = 50 Testing Data

The further increase of the frame count made acquiring data difficult. Beyond a frame count of 50, the Leap Motion controller would not have enough time to accumulate 5 intervals while the user's hand was within its hemispherical viewing area. This meant that the user's hand would have to move at a slow rate to account for this, and would not be reflective of situations in which the user may want to write moderately faster. Therefore, data is not shown for higher frame counts.

From the three frame count conditions that were tested, a frame count of 30 was optimal for our system and was used for our arm control algorithm.

Sampling Control Testing Results Summary Once these parameters were defined, the establishment of our arm control algorithm was complete. We could effectively reduce the rate at which motors react to hand movement, set a threshold velocity to exclude slow moving hands from the Leap Motion sensor, and could utilize these parameters into our motor control algorithm. These parameters could then be used to control the motors of the OWI robotic arm and examine their performance.

3.3.2 Movement About Different Axes Testing

The testing of moving different motors based on a single axis velocity will be discussed, followed by how those results led to an implementation transition.

Methods

Testing Environment This testing method is similar to the methods utilized in the sampling control testing environments. The user conducts different hand movements along a particular axis in the Leap Motion Controller's viewing range at a velocity that is above our velocity threshold. The test recognizes a change in velocity along either the x,y, or z axis and moves a specific motor of the OWI robotic arm. The group believed that this testing environment would reduce the level of interference each motor would have on another.

The OWI Robotic Arm system entails indirect physical technology. It interfaces a user with 3D technology and has a physical affect on actuators and processing effects on our software implementation. Therefore, a certain baseline observation of our robotic arm's response to the Leap Motion controller needed to be conducted quickly to asses methods of improving progress.

This is the first form of testing that would be done on a robotic arm system. In essence conducting testing on a more realized full system implementation. This is different from other motor control testing as this would now include the weight and physical capabilities of the OWI robotic arm. It was hypothesized that some previously tested performance characteristics would change, but be able to control our robotic arm effectively.

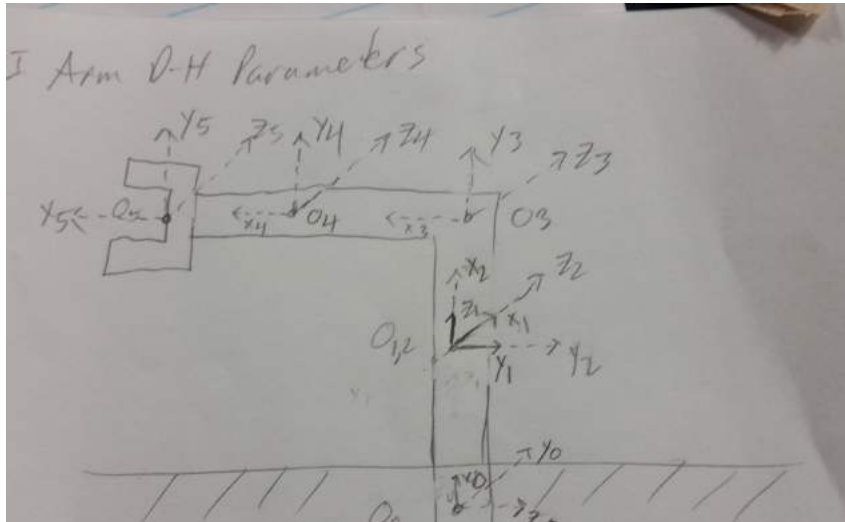


Figure 27: Denavit- Hartenberg Convention Drawing of Kinematic Chain of OWI Robotic Arm

Denavit-Hartenberg Convention for OWI Robotic Arm Figure 27 depicts the kinematic chain of the OWI robotic arm using the Denavit-Hartenberg convention. It is utilized to understand how each motor on the OWI robotic arm rotates and the division of which motors would respond to which axes. This is because with this convention, the direction of the z axis determines the axis of rotation of each joint. [32]

Two of the arm's joints are at the same location even though they are at two different locations on the physical arm. This is because in the Denavit-Hartenberg convention, the kinematic chain can be an abstraction from the physical system so long as it abides by the rules of the convention. [32]. Referring to the Denavit-Hartenberg kinematic chain in Figure 27, the DH parameters for the OWI robotic can be obtained through the process shown in below:

- α : angle between axes z_{i-1} and z_i about axis x_i to be taken positive when rotation is made counter-clockwise
- a : distance between O_i and O_{i+1}

- θ : angle between axes x_{i-1} and x_i about axis z_{i-1} to be taken positive when rotation is made counter-clockwise.
- d : coordinate of O_i along z_{i-1}

The OWI arm's DH parameters are shown in Table 8 and define the initial orientation of the OWI robotic arm.

Joint	D (in)	θ (degrees)	a (inches)	α (degrees)
Base Frame (0)	N/A	N/A	7.4	0
1	7.4	$-\pi/2$	0	0
2	0	0	9.5	0
3	0	$\pi/2$	6.5	0
4	0	0	17	0
End Effector Frame (5)	0	0	N/A	N/A

Table 8: Denavit-Hartenberg Parameters of OWI Robotic Arm

Joint-Axis Designation For joint 1 in Figure 27, it represents the base joint that contains motor 5 (as labeled in Figure 5) and its z axis is vertical, going along the height of the OWI robotic arm. This means that when this joint rotates it moves the end effector left to right. In comparison to the Leap Motion controller's axis designation, this correlates to its x axis. Therefore, this direct relationship led to motor 5 to be conditioned for rotation when a velocity exceeding 10 mm/s is detected along the x axis.

For joint 2 in Figure 27, it represents motor 4 that sits above motor 5; it is positioned at the bottom of the vertical shaft of the OWI robotic arm. Joint 3 represents motor 3 that is positioned at the top of the vertical shaft. Joint 4 represents motor 2 which is positioned behind the end effector. The z axis for all 3 of these joints are oriented in the direction of going in and out of the paper. This causes the end effector to move upward and downward. However, the end effector moving up and down corresponds to movement on Leap Motion's y axis. Therefore, these joint rotations (changes in the z axis of the robotic arm) could also change the height of the end effector to the ground. Therefore, more deliberate separation was needed.

All of the motors could not react to changes in y axis velocity for two reasons. First, it would remove particular and integral hand information about the z axis. This could create an absence of important hand information needed for gesture replication. Secondly, this would remove the amount of independence each motor would rotate in response to an axis velocity.

The group utilized the OWI controller to examine the OWI arm's physical capabilities. When motor 4 rotates, it moves the vertical shaft and the shaft the end effector rests on forward. This affects the vertical height of the end effector, but the group determined that its greater impact on the end effector position was its forward and backward displacement. Therefore, this relationship led to motor 4 to be conditioned to rotate when a velocity exceeding 10 mm/s is detected along the z axis.

When motor 3 rotates, it moves the shaft that the end effector is on and it has more of an effect on moving the end effector up and down. When motor 2 rotates, the upward and downward movement of the end effector is even greater. Thus, these relationships led to motor 2 and 3 to be conditioned to rotate when a velocity exceeding 10 mm/s is detected along the y axis. Even though two motors depend on the same axis, given only 3 axes of movement with 4 joints, there is going to be one duplicate motor condition. Motor 2 and motor 3 were chosen to be the duplicate conditions as their movement would have the least impact on the entire system's movement. Also, this was done to allow further progress and analysis of our arm control algorithm.

Testing Operations When the user's hand moves in a specific direction along a specific axis, the distance that the hand traveled is recorded. This can be obtained because Leap Motion can log the position data of the detected hand. We moved our hand in the same direction twice to get variety in

our sampling. Once this is done, the motors rotate and move the end effector. The response data that was recorded was the distance between the end effector's final location and each joints original z axis orientation, as defined by its DH parameters in Table 8. This data is logged into the motor distance section of Table 9.

The DH parameters set what the orientation of the OWI robotic arm should be before conducting a test trial. All of these parameters can be measured and once they match to what Table 8 shows, we know that the original orientation is reset and we can begin a new test trial. Since each motor rotation would have a unique effect on the end effector's position and affect a specific DH parameter, the appropriate motor can be re-oriented with the use of the OWI controller.

Further, the magnitude sign that is placed on the motor distance values is used to designate which direction the end effector moved in. This is to help distinguish the motors response for better analysis. Also, in cases where the end effector moved with less than a 0.1 inch variation from its original position, the movement was deemed negligible. This means that the end effector moved, but to an amount that was not significant.

Results Once these axes designations were made, testing data was collected and is shown in Table 9. The testing data from rotating motors based on particular axis velocities presents some inconsistencies with projected arm movement and introduces accuracy challenges.

Hand Movement Details			Motor Distance			
Axis Velocity	Distance Hand Traveled	Hand Direction	Motor 2 (Up (+)/Down (-)	Motor 3 (Up (+)/Down (-)	Motor 4 (Forward (+)/Backward (-)	Motor 5 Right (+)/ Left (-)
X-Axis	21 in	Right	0.5 in	-0.5 in	Negligible	10.5 in
X-Axis	18 in	Left	-0.1 in	-0.3 in	Negligible	-12.8 in
X-Axis	12 in	Right	0.1 in	-0.9 in	0.9 in	9.6 in
X-Axis	17 in	Left	Negligible	0.8 in	0.9 in	-11.9 in
Y-Axis	13 in	Up	0.15 in	2.4 in	Negligible	Negligible
Y-Axis	17.5 in	Down	-0.3 in	-9.5 in	-0.6 in	Negligible
Y-Axis	17 in	Up	negligible	5.5 in	Negligible	Negligible
Y-Axis	17.5 in	Down	-0.2 in	-8.4 in	0.4 in	negligible
Z-Axis	14 in	Forward	-1.2 in	Negligible	3.0 in	Negligible
Z-Axis	23 in	Backward	2 in	Negligible	Negligible	Negligible
Z-Axis	16 in	Forward	-2 in	0.2 in	2.4 in	0.6 in
Z-Axis	24 in	Backward	-0.5 in	-0.5 in	-4 in	1 in

Table 9: Movement About Different Axes Testing Data

Firstly, even though a specific motor was conditioned to move upon a specific detected velocity, all of the motors rotated and the end effector moved in reference to all of those motor rotations. Some cases were deemed negligible, but in other cases (such as z-axis hand movements), the distance between the end effector and z-axes of motor 2 and motor 4 was equivalent even though each were conditioned on different axes. Also, there are situations in which a the end effector had negligible movement even when its associated axis velocity was varying. The other form of inconsistency appears between the distance the hand traveled and how much the end effector moved in response to each motor's rotation. There is no unique scaling factor between how much the user's hand moved and how much the end effector moved.

These results imply significant levels of inaccuracy and inconsistency between the controller's detected hand characteristics and our physical system's movement. This would not allow us to set the precedent of for predictable arm movement and gesture replication. Thus, it implied the usage of forward and inverse kinematics.

3.3.3 Challenges from Testing

Forward and Inverse Kinematics The utilization of inverse kinematics is complex and requires a great amount of mathematical manipulation. Also, there are many different approaches, especially in regards to the method of geometric approaches. Bearing all of this in mind, the analysis of inverse kinematics is discussed below because it was analyzed in an attempt to further progress accurate control of the OWI robotic arm.

OWI Robotic Arm Kinematic Linkage Figure 27 illustrates the kinematic linkage of the OWI robotic arm, link length, the attached frames to the joints, and the operational axis which is along the z axis.

OWI Robotic Arm DH Parameters The Danavit-Hertenberg (DH) parameters of the OWI robotic arm robot can be extracted from the symbolic structure depicted in Figure 27. Table 8 shows the DH parameters of the OWI arm robotic arm.

Robotic Arm Forward Kinematics To find the position and orientation of the end effector of the OWI robotic arm, the derivation of forward kinematics needs to be conducted. Given the set of DH parameters of the robot, an analytic method is followed to solve the forward kinematics problem. The relationships between the position and orientation of each joint are described by the transformation matrices derived below [33]:

$${}^{i-1}_iT = R_x(\alpha_{i-1})D_x(a_{i-1})R_z(\theta_i)D_z(d_i)$$

Where, $R_x(\alpha_{i-1})$ is a rotation matrix about the x axis by α_{i-1} , $D_x(a_{i-1})$ is a translation matrix along the x axis by a_{i-1} , $R_z(\theta_i)$ is a rotation matrix along the z axis by θ_i , and D_z is the translation matrix along the z axis by d_i .

$$R_x(\alpha_{i-1}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha_{i-1} & -\sin \alpha_{i-1} & 0 \\ 0 & \sin \alpha_{i-1} & \cos \alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D_x(a_{i-1}) = \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\theta_i) = \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & 0 \\ \sin \theta_i & \cos \theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$D_z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{i-1}_iT = \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & a_{i-1} \\ \cos \alpha_{i-1} \sin \theta_i & \cos \alpha_{i-1} \cos \theta_i & -\sin \alpha_{i-1} & -d_i \sin \alpha_{i-1} \\ \sin \alpha_{i-1} \sin \theta_i & \sin \alpha_{i-1} \cos \theta_i & \cos \alpha_{i-1} & d_i \cos \alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hence, the matrix multiplication below computes the final transformation matrix that gives the position and orientation of the OWI robotic arm's end effector relative to the base frame.

$${}^0_5T = {}^0_1T {}^1_2T {}^2_3T {}^3_4T {}^4_5T$$

Let

$${}^0_5T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The position and the orientation of the end effector in roll-pitch-yaw representation is:

$${}^0_5P = \begin{bmatrix} r_{14} \\ r_{24} \\ r_{34} \end{bmatrix}$$

$$pitch = \text{Atan2}(r_{13}, \sqrt{r_{23}^2 + r_{33}^2})$$

$$roll = \begin{cases} 0 & \text{if } pitch = \frac{\pi}{2}, -\frac{\pi}{2} \\ \text{Atan2}\left(-\frac{r_{23}}{\cos(pitch)}, \frac{r_{33}}{\cos(pitch)}\right) & \text{otherwise} \end{cases}$$

$$yaw = \begin{cases} \text{Atan2}(r_{32}, r_{22}) & \text{if } pitch = \frac{\pi}{2} \\ -\text{Atan2}(r_{32}, r_{22}) & \text{if } pitch = -\frac{\pi}{2} \\ \text{Atan2}\left(-\frac{r_{12}}{\cos(pitch)}, \frac{r_{11}}{\cos(pitch)}\right) & \text{otherwise} \end{cases}$$

Robotic Arm Inverse Kinematics Utilizing inverse kinematics will allow for the designation of necessary angles that the joints must rotate in order to achieve a desired position and orientation. This is because all of the joints are rotational joints. There no prismatic joints within our system because no joints change the length of the kinematic links.

To find the joint angles of the OWI robotic arm, inverse kinematics are implemented. Since the OWI robotic arm is a 4 degree of freedom arm, a geometrical approach [33] is employed to solve the inverse kinematics problem. In the geometric approach, vectors describing the robot's state are used to calculate the joint angles of the robot.

To calculate joint 1:

Let ${}^0_G T$ be the target transformation matrix relative to the base which defines the target position and orientation.

$${}^0_G T = \begin{bmatrix} {}^0_G T_{11} & {}^0_G T_{12} & {}^0_G T_{13} & {}^0_G T_{14} \\ {}^0_G T_{21} & {}^0_G T_{22} & {}^0_G T_{23} & {}^0_G T_{24} \\ {}^0_G T_{31} & {}^0_G T_{32} & {}^0_G T_{33} & {}^0_G T_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Giving $\overrightarrow{{}_5^0 N_{K_0}} = \begin{bmatrix} {}^0_G T_{13} \\ {}^0_G T_{23} \\ {}^0_G T_{33} \end{bmatrix}$

Then, we have

$$\begin{cases} \overrightarrow{{}_5^0 P_{K_0}} = d_5 * \overrightarrow{{}_5^0 N_{K_0}} \\ \overrightarrow{{}_5^0 P_{K_0}} = \begin{bmatrix} {}^0_G T_{14} \\ {}^0_G T_{24} \\ {}^0_G T_{34} \end{bmatrix} \end{cases}$$

Which means,

$$\overrightarrow{{}_4^0 P_{K_0}} = \overrightarrow{{}_5^0 P_{K_0}} - \overrightarrow{{}_5^4 P_{K_0}} = \begin{bmatrix} {}^0_G T_{14} - d_5 * {}^0_G T_{13} \\ {}^0_G T_{24} - d_5 * {}^0_G T_{23} \\ {}^0_G T_{34} - d_5 * {}^0_G T_{33} \end{bmatrix}$$

So,

$$\theta_1 = \begin{cases} \text{Atan2}({}^0_G T_{24} - d_5 * {}^0_G T_{23}, {}^0_G T_{14} - d_5 * {}^0_G T_{13}) \\ \text{Atan2}({}^0_G T_{34} - d_5 * {}^0_G T_{33}, {}^0_G T_{24} - d_5 * {}^0_G T_{23}) + \pi \end{cases}$$

To calculate joint 3:

Let $\theta_2 = 0$,

$${}^0_2 T = \begin{bmatrix} {}^0_2 T_{11} & {}^0_2 T_{12} & {}^0_2 T_{13} & {}^0_2 T_{14} \\ {}^0_2 T_{21} & {}^0_2 T_{22} & {}^0_2 T_{23} & {}^0_2 T_{24} \\ {}^0_2 T_{31} & {}^0_2 T_{32} & {}^0_2 T_{33} & {}^0_2 T_{34} \end{bmatrix}$$

$$\overrightarrow{{}_2^0 P_{K_0}} = \begin{bmatrix} {}^0_2 T_{14} \\ {}^0_2 T_{24} \\ {}^0_2 T_{34} \end{bmatrix}$$

Thus,

$$\begin{aligned}\overrightarrow{{}_2P_{K_0}} &= \overrightarrow{{}_4P_{K_0}} - \overrightarrow{{}_2P_{K_0}} = \begin{bmatrix} \overrightarrow{{}_2P_{K_0x}} \\ \overrightarrow{{}_2P_{K_0y}} \\ \overrightarrow{{}_2P_{K_0z}} \end{bmatrix} \\ \phi &= \text{Asin} \left(\frac{\left(l_1^2 - a_2^2 + \left| \overrightarrow{{}_2P_{K_0}} \right|^2 \right)}{2 \left| \overrightarrow{{}_2P_{K_0}} \right| l_1} \right) + \text{Asin} \left(\frac{\frac{\left| \overrightarrow{{}_2P_{K_0}} \right|^2 - l_1^2 - a_2^2}{2 \left| \overrightarrow{{}_2P_{K_0}} \right|}}{a_2} \right) \\ \alpha &= \text{Atan2}(-d_4, a_3)\end{aligned}$$

So,

$$\theta_3 = \begin{cases} \pi - \phi - \alpha \\ \pi + \phi - \alpha \end{cases}$$

To calculate joint 2:

$$\begin{aligned}\overrightarrow{{}_4P_{K_2}} &= {}_2R \overrightarrow{{}_4P_{K_0}} = {}_2R^{-1} \overrightarrow{{}_2P_{K_0}} \\ {}_2T &= \begin{bmatrix} {}_2^0R & {}_2^0P_{ORG} \\ 0 & 1 \end{bmatrix} \\ {}_2R &= \begin{bmatrix} {}_2^0T_{11} & {}_2^0T_{12} & {}_2^0T_{13} \\ {}_2^0T_{21} & {}_2^0T_{22} & {}_2^0T_{23} \\ {}_2^0T_{31} & {}_2^0T_{32} & {}_2^0T_{33} \end{bmatrix} = {}_2R^{-1} \\ \overrightarrow{{}_4P_{K_2}} &= \begin{bmatrix} {}_2^0T_{11} & {}_2^0T_{12} & {}_2^0T_{13} \\ {}_2^0T_{21} & {}_2^0T_{22} & {}_2^0T_{23} \\ {}_2^0T_{31} & {}_2^0T_{32} & {}_2^0T_{33} \end{bmatrix} \begin{bmatrix} \overrightarrow{{}_2P_{K_0x}} \\ \overrightarrow{{}_2P_{K_0y}} \\ \overrightarrow{{}_2P_{K_0z}} \end{bmatrix}\end{aligned}$$

Thus,

$$\begin{aligned}\beta_1 &= \text{Atan2} \left(\overrightarrow{{}_4P_{K_0x}}, \overrightarrow{{}_4P_{K_0y}} \right) \\ \beta_2 &= \text{Asin} \left(\frac{\left(a_2^2 - \left| \overrightarrow{{}_4P_{K_0}} \right|^2 + l_1^2 \right)}{2 l_1 a_2} \right) + \text{Asin} \left(\frac{l_1 - \frac{a_2^2 - \left| \overrightarrow{{}_4P_{K_0}} \right|^2}{2 l_1}}{\left| \overrightarrow{{}_4P_{K_0}} \right|} \right)\end{aligned}$$

Therefore,

$$\theta_2 = \begin{cases} \frac{\pi}{2} - (|\beta_1| + \beta_2) \\ \frac{\pi}{2} + (|\beta_1| - \beta_2) \end{cases}$$

To calculate joint 4 and joint 5:

$${}^4_5R = {}^0_4R^{-1} {}^0_5R = {}^4_0R {}^0_5R$$

$${}^4_5R = Rot_z(\theta_4)Rot_y(\theta_5)$$

In which,

$$Rot_z(\theta_4) = \begin{bmatrix} \cos \theta_4 & -\sin \theta_4 & 0 \\ \sin \theta_4 & \cos \theta_4 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$Rot_y(\theta_5) = \begin{bmatrix} \cos \theta_5 & 0 & \sin \theta_5 \\ 0 & 1 & 0 \\ -\sin \theta_5 & 0 & \cos \theta_5 \end{bmatrix}$$

$${}^4_5R = \begin{bmatrix} {}^4_5R_{11} & {}^4_5R_{12} & {}^4_5R_{13} \\ {}^4_5R_{21} & {}^4_5R_{22} & {}^4_5R_{23} \\ {}^4_5R_{31} & {}^4_5R_{32} & {}^4_5R_{33} \end{bmatrix}$$

Therefore,

$$\theta_4 = Atan(-{}^4_5R_{31}, {}^4_5R_{33})$$

$$\theta_5 = Atan(-{}^4_5R_{12}, {}^4_5R_{22})$$

Additional Sensors/components Even with inverse kinematics analysis and coding the geometric approach method for solving inverse kinematics equations, there would be challenges in translating these values to the system. This is because the evaluated angles would only determine the joint angles in reference to the entire OWI robotic arm system and cannot be translated to its DC motors.

DC motors have a continuous axis of rotation. This is in contrast to servo motors, which have a specified range of angular rotation (ranging from 60 to 180 degrees). Therefore a direct translation of the evaluated joint angles to the servo angle motor can be made. An external component or system would be needed in order to translate appropriate DC motor rotation to a specified angle.

It was postulated that the utilization of sensors with the Adafruit clone motor shield would be able to aid our control of the OWI robotic arm. This was inspired by another OWI robotic arm implementation. The project in [34] utilizes the analog sensor inputs of the Adafruit motor shield to attach potentiometers to the physical structure of the robotic arm (as shown in Figures 28, 29, and 30).

These potentiometers have metal rods connecting them to other portions of the arm. This is done so that as the motors the potentiometers will rotate. This varying resistance is then read by the motor shield's the analog inputs of the motor shield and transferred into the Arduino microcontroller. It was hypothesized that the each joint's rotation could be approximated by correlating the joint angle from inverse kinematics to a potentiometer reading.



Figure 28: 1st Example of Connecting Potentiometer to OWI Arm [34]

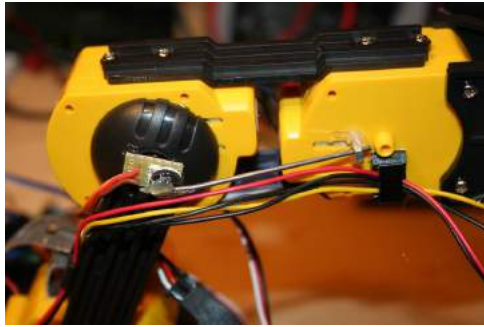


Figure 29: 2nd Example of Connecting Potentiometer to OWI Arm [34]



Figure 30: 3rd Example of Connecting Potentiometer to OWI Arm [34]

Using a low tolerance linear potentiometer, the approximate slope of the resistance would be linear. In which case the slope intercept formula could solve for either unknown. Dividing the range of potentiometer values into separate sections could account for variations in slope across its taper. The Arduino microcontroller could then be programmed such that it will rotate the OWI's motors and continuously read the resistance value until the specified potentiometer value has been reached. Achieving this would be able to give a relationship between inverse kinematics and the physical structure of the OWI robotic arm.

However, implementation introduced several challenges. First, a large potentiometer value needed to be used in order to account for a wide range of angles. If the potentiometer variance was too small, then small deviations between corresponding angle values would require a very precise potentiometer value. Secondly, specially designed trim pots were needed in order to facilitate the mechanical structuring as shown in the previous figures. This made it difficult to have quick access to potentiometers that both matched our physical preferences and be a large enough value. Thirdly, there were difficulties in mechanically fixating the pots onto the robotic arm structure. Long drying times of glue elongated this process and metal rods not allowing proper rotation of the motors, or breaking off during movement exacerbated this problem. These challenges prevented further experimentation with these sensors.

OWI Testing Summary Utilizing testing procedures to aid in the evaluation of the sampling control parameters aided in the development of the arm control algorithm and improved control of motor response to hand movement. However, testing for motor rotation reacting to movement along different axes introduced difficulties to progress further analysis and development.

The motor control algorithm proposed the potential for accurate control of a motor responding to specific axis velocities. If a system could be implemented in which the rotation of one motor can be determined based on a single axis velocity and still produce accurate writing and replication, this algorithm could be used as the foundation for that system. This postulation is what inspired the group to utilize a new implementation, mainly involving the XY Plotter.

4 XY Plotter Robotic System

4.1 Background

4.1.1 Introduction to XY Plotter



Figure 31: Picture of XY plotter [35]

The XY Plotter is a drawing robot that can move a pen or other instrument to draw digital artwork on a flat surface. It has two stepper motors and each of them can control one axis (X or Y) of movement. It has another servo motor to lift the pen up and down. The picture of the XY plotter can be seen in Figure 31. As mentioned in the previous section, the servo motor robotic arm and OWI robotic arm do not meet our functionality requirements. The XY Plotter is the third robotic system we tried to use and, compared with the other two robotic arms, it has several advantages.

- Enough DOF for writing: When writing, we only need 3 DOF for linear movement, description of roll-pitch-yaw angles is not necessary.
- Reduce complexity of inverse kinematics: The XY Plotter has two stepper motors to control the x and y direction separately, which means that inverse kinematics can be implemented easier because of reduced interaction between two motors.

- Closer to real hand writing gestures: Even when we use our hand to write our signature, we do not rotate our elbow or frequently move our arm. Most of time, we use our fingers to hold the pen and let it move in the x-y plane while keeping our arm stationary. Therefore, the X-Y plotter more accurately replicates the movement of the hand when humans write.

4.1.2 Main Components of XY Plotter Robotic System

Stepper Motor

Stepper Motor A stepper motor is a brushless DC electric motor that divides a full rotation into a number of equal steps. The stepper motors the XY Plotter use are 42BYG stepper motors. They are two-phase four-wire stepper motors and the step angle is 1.8deg per step. The picture of one 42BYG stepper motor and the wiring diagram is shown in Figure 32, 33. Since the step angle of 42BYG is 1.8 deg per step, it is accurate enough for hand writing.



Figure 32: Picture of 42BYG Stepper Motor [36]

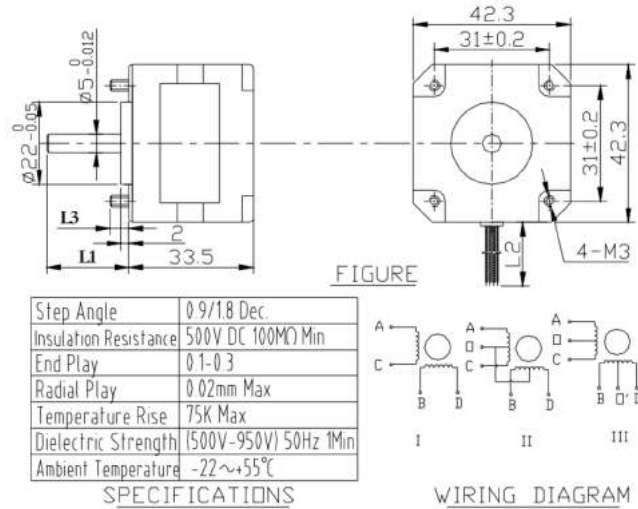


Figure 33: The Wiring Diagram of 42BYG [37]

AdvancedFirmata In order to control a stepper motor with Arduino via a computer, we need to use a stepper motor driver called A4988 (discussed in the next section) and upload our Arduino with libraries called configuredFirmata and AdvancedFirmata. Using these libraries, we can control the stepper motor through a laptop. [38]

The Stepper Motor Driver Unlike a DC motor or servo motor that can be controlled easily with a few wires, a stepper motor generally requires a more involved hardware setup, requiring a stepper motor driver. Even though the XY Plotter came packaged with a motor driver, using it would require additional components and connections. The A4988 is an ideal stepper motor driver for our system because there is available documentation on this stepper motor driver's usage in the Johnny-Five framework [39]. The wiring diagram of the A4988 is shown in Figure 34.

In order to control this chip, we need two power sources. One is a 8-35V supply that is used to provide power to the step motors and the other one is a 3-5V supply used to enable the chip. We need four wires to control these two-phase four-wire stepper motors and two I/O pins to connect to the Arudino board. This controls the direction and steps of the stepper motors. Lastly, for proper operation, the enable pin must be enabled and the reset pin must be disabled. The wiring in Figure 34 is the simplest way to configure this stepper motor driver. If we want our stepper motor to run in different states, we need to set the rest of the pins as well.

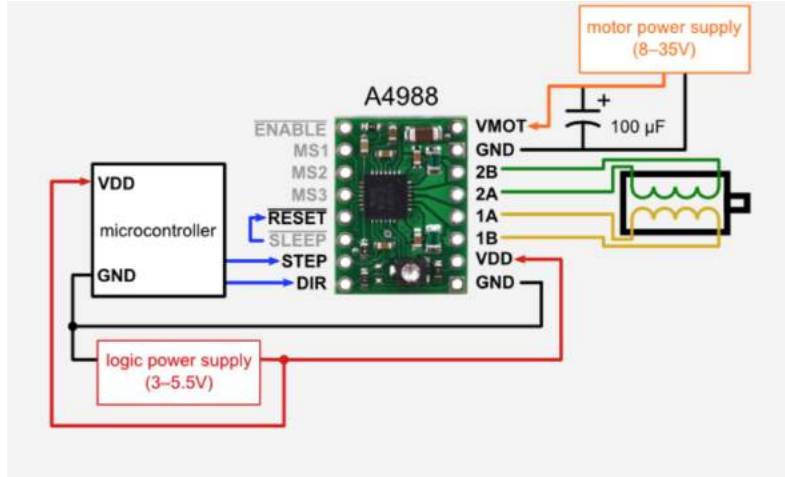


Figure 34: Picture of A4988 Pin Diagram[39]

Servo motor A servo motor is a rotary actuator or linear actuator that allows for precise control of angular or linear position. The servo motor is used to lift the pen up and down. It can be easily implemented with an Arudino's PWM pin.



Figure 35: Picture of Servo Motor [40]



Figure 36: Picture of Me Orion [41]

Me Orion Makeblock Orion is an easy-to-use mainboard based on Arduino Uno with improvements for education. It provides eight RJ25 ports to connect all the other Me series modules with color-labels, but it is not supported by Johnny-Five. Therefore we decided to use the Arduino Uno instead, which means a separate circuit for connection is needed (which will be discussed later). Figure 36 shows the Me Orion board and the only part we used from it is the 12V source that the stepper motors need. [41]

Micro Switch Button Every side of the XY Plotter has one micro switch button. These buttons will be pressed by the end effector when the stepper motors move to an edge of the XY Plotter. It is an easy-control three-leg button, and once the button is pressed down, it will send a signal to the micro-controller to stop the stepper motors.

4.1.3 Supplementary components of XY Plotter Robotic System

To supplement the XY Plotter as a solution to accomplishing our project goal, additional components to assist functionality are necessary.

LCD Display The purpose of the LCD screen is to display significant data or signals while running, testing or debugging our code. The LCD screen we used is Grove RGB Backlight 16x2 LCD and uses an I2C communication bus. I2C is an easy-to-use communication bus with 4 wires, namely VCC, GND, SDA and SCL. The LCD screen useful for debugging.



Figure 37: Picture of LCD

Glass Ergonomic Tabletop Riser The purpose of this component is to provide a horizontal surface for the user to write/draw. This removes displacement in the vertical direction while writing above the Leap Motion. It was hypothesized that this would make the writing process more stable and smooth.

However, after multiple tests we realized that the accuracy of Leap Motion's hand movement detection degraded when the glass surface was placed above the sensor. This produces unstable results. Therefore, we decided to go back to writing in the air.



Figure 38: Picture of Glass Surface [42]

Touch Sensor: When people write their signature, the pressure of the writing tool to the surface varies. The purpose of the touch sensor is to detect the pressure when we write. In order to use the touch sensor, we need to attach it to the user's finger. We also need one analog input to get the signal. Based on varying pressure, the Arduino will get different values of voltage and then let the servo motor turn different angles to change the pressure on the pen. In this way, we can successfully mimic the varying pressure of writing. However, it was not used because our a glass table reduced accuaracy and we have to write our signature in the air. This means that the touch sensor cannot detect pressure.



Figure 39: Picture of Force Sensitive Resistor (Touch Sensor)

Ultra-Thin Dry Erase Whiteboard Before choosing Ultra-thin Dry Erase Whiteboard as our writing/painting carrier, we used letter-size paper to examine the system's writing capability. However, we found some serious drawbacks during testing with using this type of paper; including inaccurate writing and replication results and excessive paper waste. We will talk more reasons in testing/verification or debugging section.

The Ultra-Thin Dry Erase Whiteboard is more stable and smooth compared with letter-size paper, resulting in more precise and accurate writing/replication. In addition, because it is erasable, it reduces paper and waste and is more environmentally conscious.



Figure 40: Picture of Ultra-thin Dry Erase Whiteboard [43]

4.1.4 Overview of XY Plotter Robotic System

Since our users write their signature in the air, it is difficult for them to know what they are writing. As mentioned before, we tried different solutions, such as including a glass table to our system. Eventually, a user interface was created that can track hand movement and display it on a screen. This facilitates easier usability and monitoring.

User Hand Motion Simulation Interface

Leap Motion Visualizer At first, we wanted to use the visualizer provided by Leap Motion device (shown in Figure 41). This visualizer provides a lot of data such as the speed of the x, y, and z directions, the current position of your hand and many more. However, we found the visualizer to not be programmable. In other words, since it did not provide us functions for displaying hand writing motion trajectory, it is not optimal for our system.

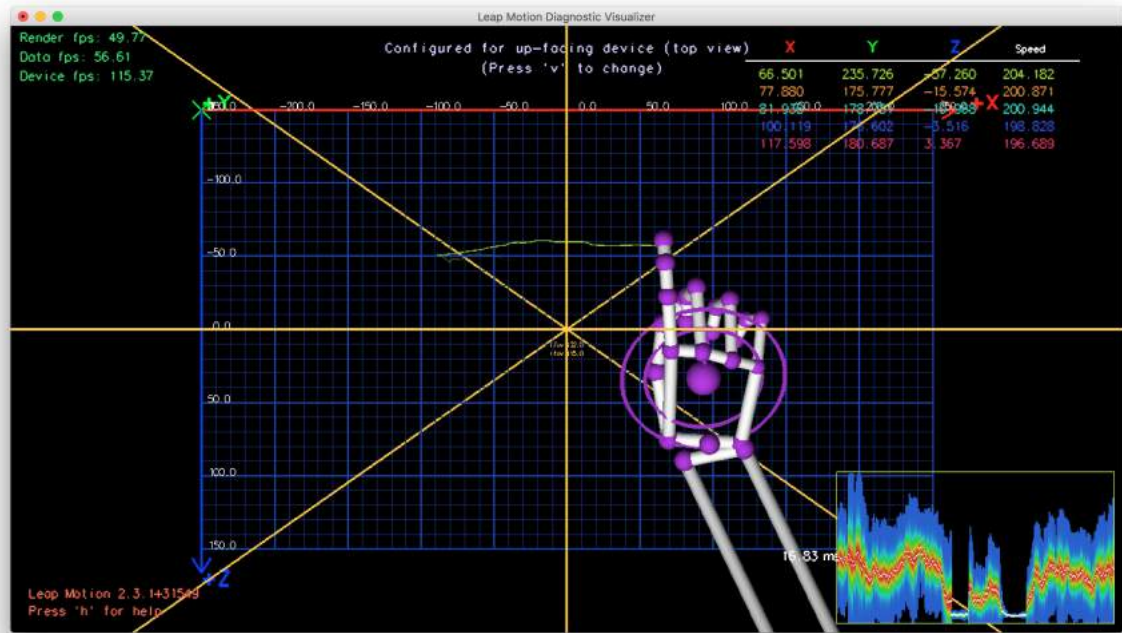


Figure 41: Picture of Visualizer

Finger Trajectory Tracker Fortunately, there is an open source website called CodePen [44] where we can program HTML and JavaScript code and implement real-time hand motion tracking by using Leap Motion (Figure 42). When we write our team members' signatures with our XY Plotter system, the results also appear in the user interface on the computer screen (shown in Figure 43, 44, 45 and 46). With this user interface, the user can watch the monitor and better observe what he/she is drawing or writing in the air above Leap Motion. Our open-source coded website could be checked and used here: <https://codepen.io/Seasean/pen/GNQdBY/>

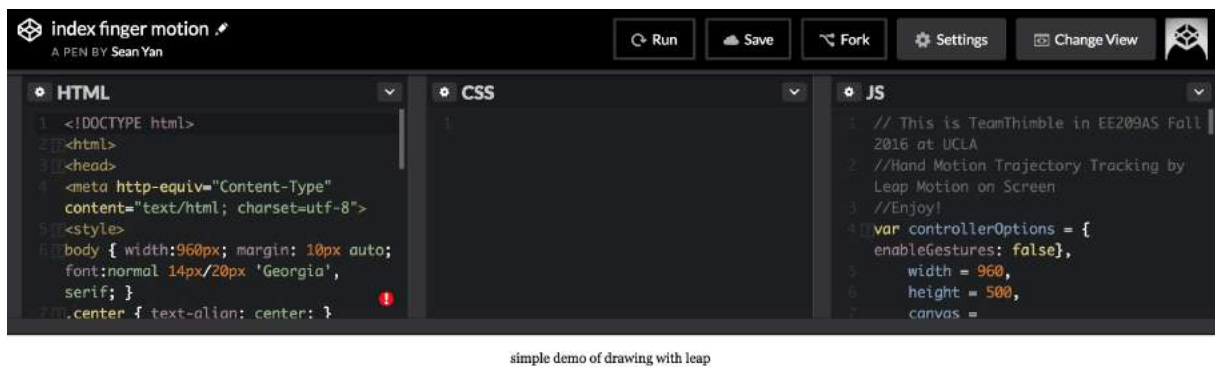


Figure 42: Picture of Editor of Finger Trajectory Tracker [44]

Also, we implemented functions on this tracker so that it has the same hand gestures as the XY Plotter algorithms for recording arrays and writing (described in writing and replication parts). This would ensure that the trajectory on the tracker correlates to hand movement while writing with the XY Plotter.

In addition, before drawing or writing, no writing trajectory will be displayed on the monitor. Only a slight spot trajectory will appear to indicate the location of the user's index finger. Once a user is ready to write, the slight spot trajectory will disappear, and the blue spot indicating writing motion will appear on the screen.

Finally, when a user no longer wants the trajectory to be shown on the monitor, or they accidentally made mistakes while drawing, the user can swipe their palm quickly (at a horizontal velocity larger 1000 mm/s). This clears the demo screen and is an easier way to refresh the screen than manually clicking a button.

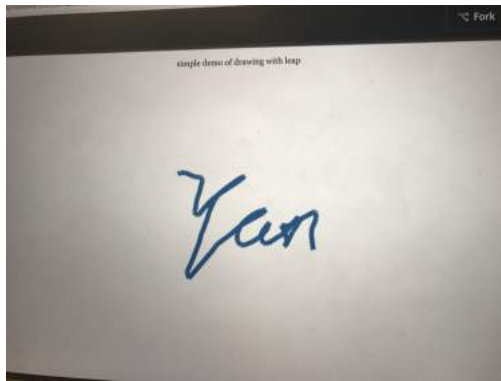


Figure 43: "Yan" Signature on Finger Trajectory Tracker

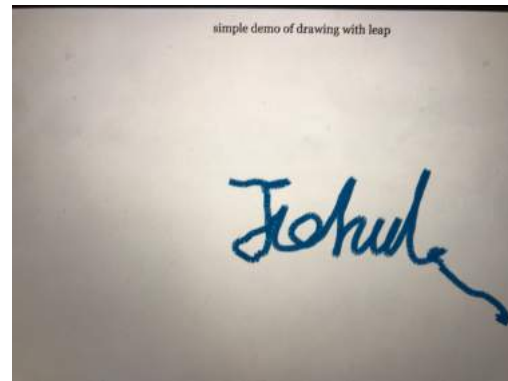


Figure 44: "Jiahui" Signature on Finger Trajectory Tracker

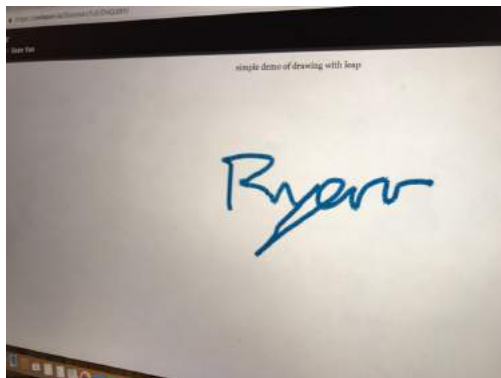


Figure 45: "Ryan" Signature on Finger Trajectory Tracker



Figure 46: "alia" Signature on Finger Trajectory Tracker

XY Plotter Robotic System Diagrams The schematic of our system can be seen in Figure 47. We acquired the data from Leap Motion and then the data is processed by our laptop. After processing the data, the laptop sends a signal to the Arduino to control the movements of stepper motors, servo motor and other actuators. At the same time, the writing trajectory will be shown in the user simulation interface as well. We use two different power adapters: 12V to power the stepper motors and 5V to power our system. The whole system is shown in Figure 48.

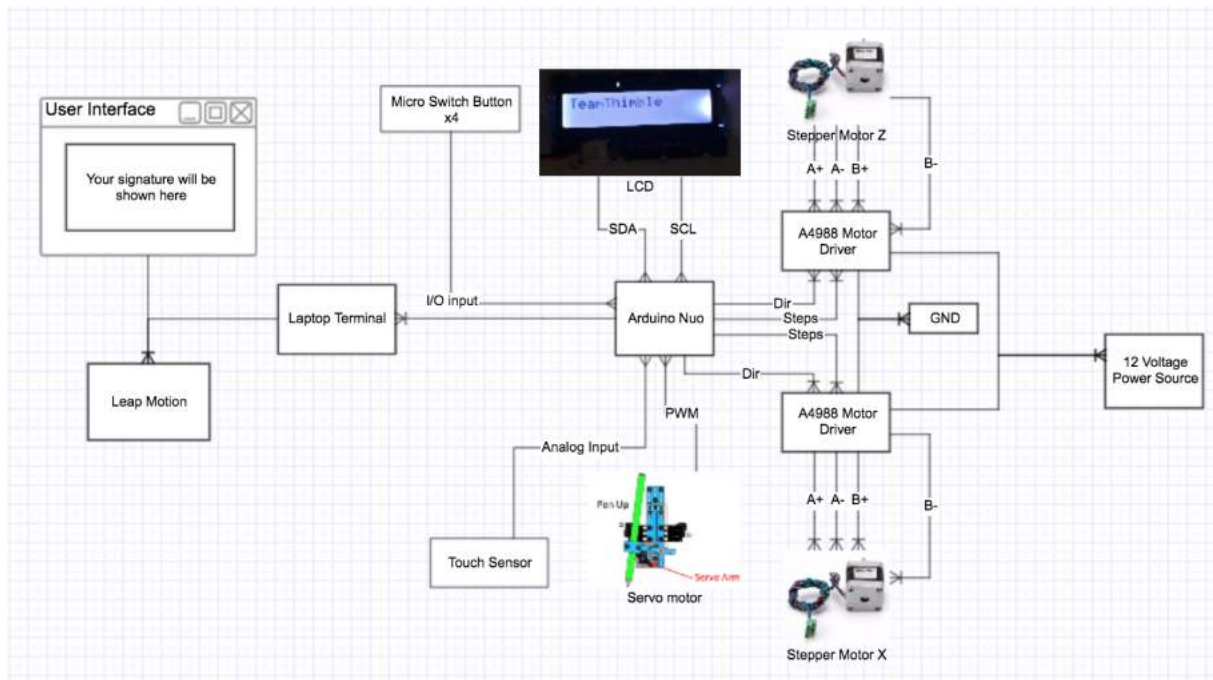


Figure 47: Picture of System Schematic

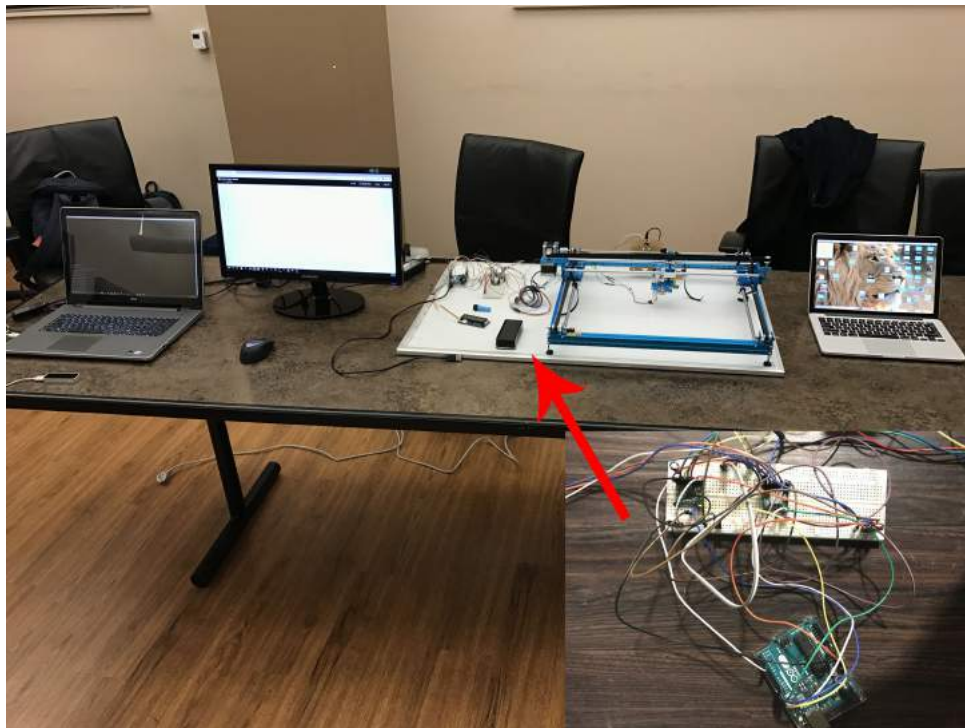


Figure 48: Whole System of XY Plotter Robotic System

4.2 Writing Process Methods

Pre-processing

Coordinated System Setting Figure 48 depicts the entire XY Plotter system that was built. It still requires the creation of an algorithm to control it. Before we write our algorithm, we need to define

our own coordinate system for the XY Plotter. The coordinate system of Leap Motion is shown in Figure 1. In order to control it more intuitively and clearly, (see Figure 49), we set the direction of x axis on the XY Plotter to be the same as that of Leap Motion's, and let the y axis of the XY Plotter have the same direction as the z axis of Leap Motion. This distinguishes the axes of the XY Plotter and indicates that variations along Leap Motion's y axis will not rotate the stepper motors on the XY Plotter. Also, this coordinate system will be easier to control with Arduino Nuo as well.

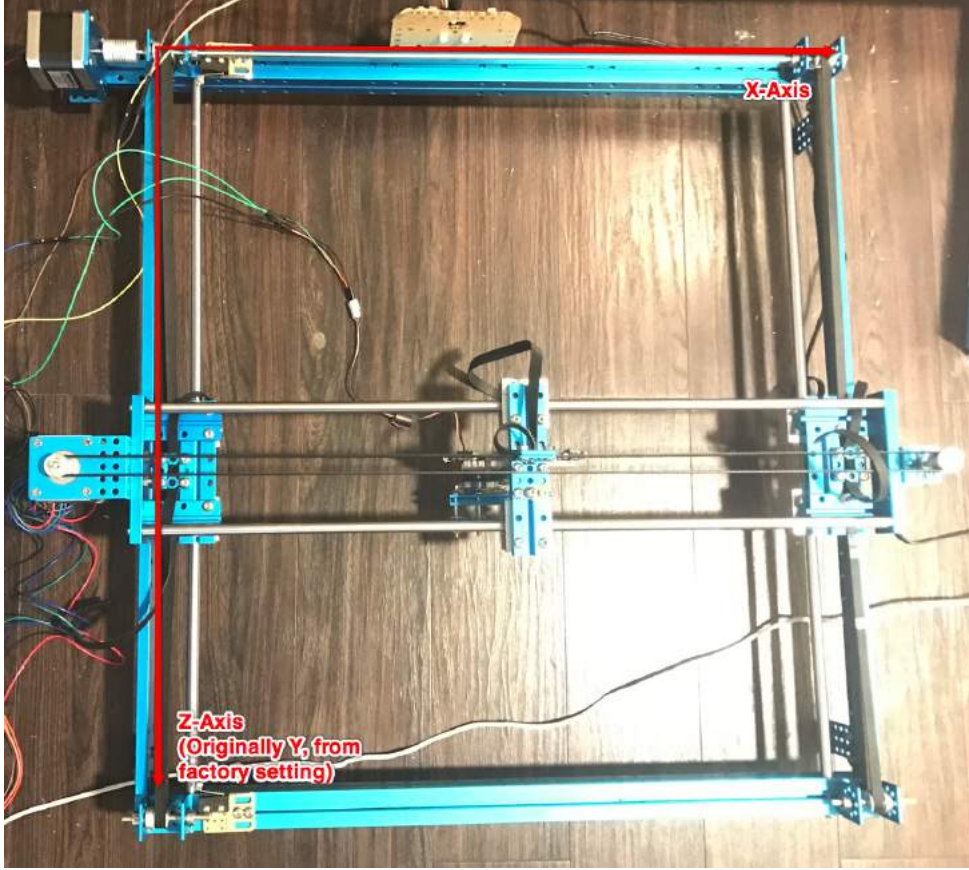


Figure 49: Coordinate System of XY Plotter

Length Measurement In order to quantitatively measure the exact distance we write, we need to know how long is one step of stepper motor. After analyzing our robotic system, we found that the circumference of the rod that connects the belt and stepper motor is 37.7 mm. In other words, once the stepper motor moves one step, the belt will move $37.7 * 1.8/360 = 0.1885mm$ (We define $\alpha = 1/0.1885 \approx 5$). α is an important factor in our algorithm because it transfers rotation into linear motion.

Frame Testing Since Leap Motion's sample frequency is around 40 FPS, each frame duration is 25 ms. If we set the speed of the stepper motors to 120 rpm (0.002 rad per millisecond), after every millisecond the stepper motors move 0.4 steps. This means at most, in one frame, the stepper motors can move 10 steps.

4.2.1 XY Plotter Writing Process Algorithm

Speed-Only Control Algorithm

Algorithm Introduction The first hypothesized idea that was implemented was using speed control. The basic idea of this algorithm is functionally similar to the sampling control used in the OWT's arm control algorithm. The utilization of hand motions that control motor rotation, direction, and stoppage is duplicated in this algorithm. The testing results of this algorithm that depict its inaccuracy are described in the XY Plotter section titled Results.

Speed-Position Control Algorithm

Algorithm Introduction This algorithm aims to improve upon the first algorithm. In this algorithm, we add two variables. One is the speed ratio that is used to determine which stepper motor controls most of the motion within a frame. The other is the relative position between two sampling points that is used to determine how many steps the stepper motors should move. More accurate results are attainable with this algorithm, but it has difficulties with drawing curves (as described in XY Plotter Results).

Bresenham's Line Algorithm

Algorithm Introduction The Bresenham line Algorithm is an algorithm that can be used to write a curve. Bresenham demonstrates a line based algorithm that can be used to write a curve with a digital plotter [45]. The basic theory is to split the writing area into 8 octants shown in Figure 50. Every part has two movements. For example, any line that is within Area 1 that has the same orientation as Line 1 can be described with movement M_1 or M_2 . We sampled the curve sufficiently enough so that a curve can be approximated with differential linear segments. Therefore the segment between D_1 and D_2 in Figure 51 is approximated as having a linear slope.

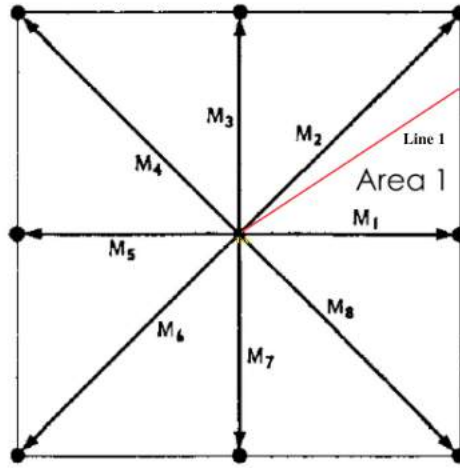


Figure 50: Bresenham 8 Octant Writing Area [45]

The smallest path the digital plotter can move is one unit, which is equivalent to one step in the XY Plotter. In order to draw D_1D_2 (shown in Figure 52), the distance from R to D_1D_2 and the distance from Q to D_1D_2 are both calculated. If the former is shorter, then the plotter will move along M_1 . If the latter is shorter, then the plotter will move along M_2 . We can follow this process sequentially until the path from D_1 to D_2 is determined.



Figure 51: Sample Curve [45]

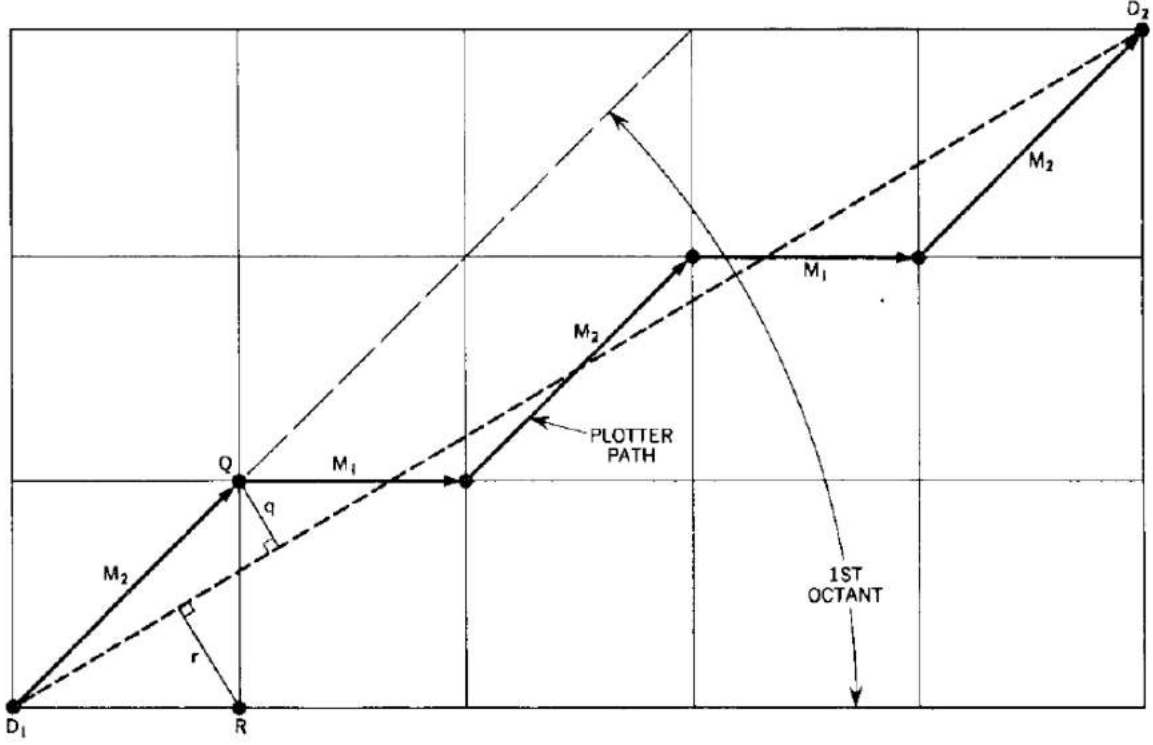


Figure 52: Path from D_1 to D_2 [45]

Modified Bresenham's Line Algorithm

Algorithm Introduction This is a modification of Bresenham's original line algorithm. In this algorithm, we separate our writing area into 4 quadrants instead of 8 octants (reduce computational load). With a fast sampling frequency of 40 FPS (25 ms/frame) and an accuracy of 0.2 mm for the length per step, the stepper motors controlling the x axis and z axis movement can rotate at the same time. This is because its improved accuracy reduces the amount of deviation each step takes from the straight line approximation. In reference to the original Bresenham's Line Algorithm, to determine the initial and final point between two consecutive frames, the relative position of the x axis and z axis are considered. Their direction can be determined with the ratios between the x axis velocity and z axis velocity.

4.3 Writing Process Testing

4.3.1 Speed-Only Control Algorithm

Methods Similar to the testing method for the OWI robotic arm, a velocity threshold and frame count parameter needed to be determined in order to utilize this algorithm.

Results After testing several different speeds, we found ± 50 mm/s is the optimal threshold for rotation direction of the x axis and z axis stepper motors. After several tests of frame count optimization, the writing was most legible when the frame count was set between 5 and 10. In order to increase the accuracy, we control the stepper motor every five frames and if the magnitude of the speed of all five frames is larger than 50 mm/s, the stepper motor will move (shown in 10).

	>50 mm/s	$-50 < v < 50$ mm/s	>50 mm/s
Stepper Motor X	10	0	-10
Stepper Motor Y	10	0	-10

Table 10: Speed-Only Control Algorithm Results

The signature we get from the XY Plotter is shown in Figure 53 and 54. The speed-only algorithm allowed the XY plotter to successfully track the movement of our hand. We can see from Figures 53 and 54 that the results are not reflective of what was written by hand (the writing at the bottom of Figure 53).

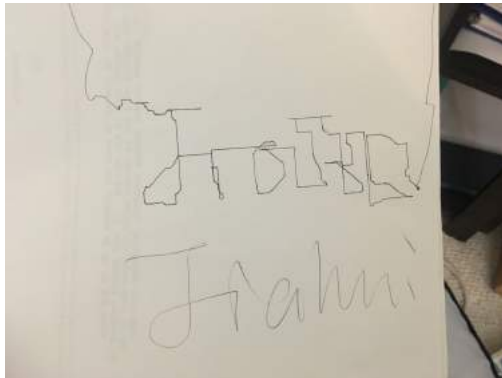


Figure 53: First Version of Signature

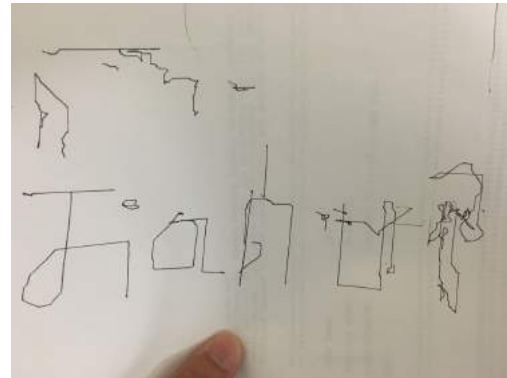


Figure 54: Second Version of Signature

These results demonstrate a particular problem with this algorithm. Suppose we want to write a line from A to B (shown in Figure 55). As an ideal example, when our hand moves, one velocity (v_x) is larger than 50 mm/s and the other velocity (v_z) is 0, and vice versa. However, realistically v_x is larger than 50mm/s and sometimes v_z (represented by ySpeed in Figure 56) is larger than 50mm/s as well.

This problem displays the disparity between the expected and real situation. There is a big leap from point B to point C which results in the common inaccurate representation of the letters "h" and "i", as shown in figures 53 and 54. This problem could be solved by considering the utilization of other data provided by Leap Motion.

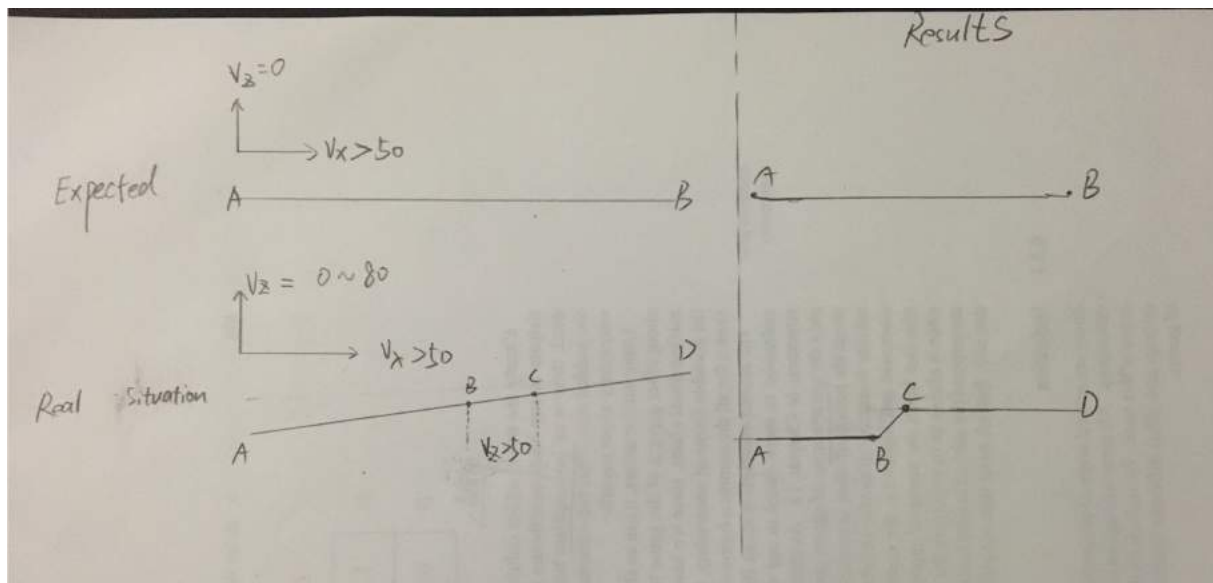


Figure 55: The Comparison Between Expected and Real Movement

```

finalPrj -- -bash -- 120x40
x=> 32.8425 xSpeed=> 251.85 y=> 0.806199 ySpeed=> -328.301
x=> 34.9714 xSpeed=> 252.452 y=> -0.632046 ySpeed=> -285.647
x=> 37.1723 xSpeed=> 255.309 y=> -0.998301 ySpeed=> -219.992
x=> 39.8833 xSpeed=> 270.63 y=> -0.987007 ySpeed=> -157.988
x=> 42.0168 xSpeed=> 265.477 y=> -0.760995 ySpeed=> -107.53
x=> 44.7128 xSpeed=> 278.004 y=> -0.594522 ySpeed=> -71.7688
x=> 47.1657 xSpeed=> 280.371 y=> -0.0409439 ySpeed=> -33.9796
x=> 49.5796 xSpeed=> 280.383 y=> 0.510184 ySpeed=> -6.97101
x=> 51.5069 xSpeed=> 266.111 y=> 1.54696 ySpeed=> 28.2374
x=> 53.1314 xSpeed=> 244.59 y=> 2.43851 ySpeed=> 49.574
x=> 55.0205 xSpeed=> 237.733 y=> 3.30169 ySpeed=> 64.2298
x=> 57.1194 xSpeed=> 239.609 y=> 4.01739 ySpeed=> 70.0977
x=> 59.6019 xSpeed=> 250.599 y=> 4.73126 ySpeed=> 73.53
x=> 61.4691 xSpeed=> 243.383 y=> 5.78719 ySpeed=> 87.1913
x=> 63.6493 xSpeed=> 245.961 y=> 6.07899 ySpeed=> 72.7632
x=> 66.4683 xSpeed=> 267.91 y=> 6.55318 ySpeed=> 67.7267
x=> 68.7363 xSpeed=> 267.015 y=> 6.74809 ySpeed=> 54.5721
x=> 71.1009 xSpeed=> 268.705 y=> 7.124 ySpeed=> 50.8418
x=> 73.4013 xSpeed=> 267.997 y=> 7.34427 ySpeed=> 44.2765
x=> 75.5505 xSpeed=> 264.005 y=> 7.46443 ySpeed=> 35.8299
x=> 77.3723 xSpeed=> 249.153 y=> 7.35102 ySpeed=> 23.007
x=> 79.1702 xSpeed=> 237.763 y=> 7.56512 ySpeed=> 23.6141
x=> 81.6677 xSpeed=> 249.895 y=> 7.68851 ySpeed=> 21.667
x=> 83.1166 xSpeed=> 225.516 y=> 7.75971 ySpeed=> 18.9267
x=> 83.2891 xSpeed=> 166.927 y=> 7.76626 ySpeed=> 14.2854
x=> 82.5046 xSpeed=> 96.6638 y=> 7.98097 ySpeed=> 17.7052
x=> 81.9291 xSpeed=> 51.4684 y=> 7.88499 ySpeed=> 10.512
x=> 81.9717 xSpeed=> 35.9721 y=> 7.73432 ySpeed=> 3.35285
x=> 81.2797 xSpeed=> 4.58879 y=> 8.89396 ySpeed=> 39.1116
x=> 81.8215 xSpeed=> 19.0543 y=> 8.8459 ySpeed=> 26.4065
x=> 82.8966 xSpeed=> 45.2293 y=> 9.81751 ySpeed=> 48.2687
x=> 84.189 xSpeed=> 69.9562 y=> 9.86084 ySpeed=> 35.2764
x=> 85.0122 xSpeed=> 72.2209 y=> 9.68385 ySpeed=> 19.471
x=> 86.3 xSpeed=> 87.8376 y=> 10.4001 ySpeed=> 35.1328
x=> 87.7785 xSpeed=> 103.368 y=> 11.0901 ySpeed=> 47.7241
x=> 89.4969 xSpeed=> 120.566 y=> 12.1289 ySpeed=> 68.744
x=> 91.2644 xSpeed=> 132.653 y=> 12.0291 ySpeed=> 51.3833
x=> 93.6655 xSpeed=> 159.664 y=> 12.1722 ySpeed=> 40.7419
^C
lijiiahuis-MacBook-Pro:finalPrj lijiiahuis$

```

Figure 56: The Testing Data From Terminal

4.3.2 Speed-Position Control Algorithm

Methods In this algorithm, the velocity threshold of ± 50 mm/s is used as well as new variables called ratio 1_3 (which is the rate of v_x over v_z) and ratio 3_1 (which is the rate of v_z over v_x). These are calculated to find develop an even more suitable threshold for allowing the stepper motors to move.

With these defined parameters, we want this algorithm to operate such that When v_x is larger than 50 mm/s and ratio 1_3 is larger than a certain value, the stepper motor x will move. When v_z is larger than 50mm/s and ratio 3_1 is larger than a certain value, the stepper motor z will move. The value used was 0.5, which is further explained in the next section.

Results After multiple testing trials were done, certain trends were observed. If the threshold factor is too small, the possibility of two motors moving together, which greatly reduces functionality, increases. If the threshold factor is too large, then the possibility of only one motor moving increases.

Ratio 1_3 and Ratio 3_1 are bounded between 0 and 1. This is because if the threshold is 0, this algorithm functions the same way as speed-only algorithm. If the threshold is 1, it becomes difficult for both motors to move and it is difficult to draw curves. This increased limitation makes the writing appear more square like. Therefore, the suitable ratio was found to be 0.5.

Another improvement we have made is to let stepper motor move a different amount of steps based on the relative motion of our hand. We recorded every position of our hand and calculated the difference between the last and current hand position, called error position (errX for example). After multiplying by the factor α , we round numbers of $\alpha * errX$, and set it as the input of the stepper motor.

Stepper Motor	Ratio	$>50 \text{ mm/s}$	$-50 < v < 50 \text{ mm/s}$	$>50 \text{ mm/s}$
Stepper Motor X	Ratio1-3 >0.5	$\alpha * errX$	0	$\alpha * errX$
	Ratio1-3 <0.5	0	0	0
Stepper Motor Z	Ratio3-1 >0.5	$\alpha * errZ$	0	$\alpha * errZ$
	Ratio3-1 <0.5	0	0	0

Table 11: Speed-Position Control Algorithm

Results With these two improvements, we get the results shown on Figure 57 and 58. Clearly, the results are better than the results obtained by speed-only algorithm. However this algorithm could use further improvement. Figures 57 and 58 show that the letters written are square-like in appearance. In other words, it is hard for this algorithm to write a curve because of the ratio variables. This is because at the initial drawing of a curve, one motor will not move even though it should move slowly (in order to create the downward motion). This means that the algorithm only produces straight line segments. The result of drawing a curve with this algorithm is shown in Figure 59.

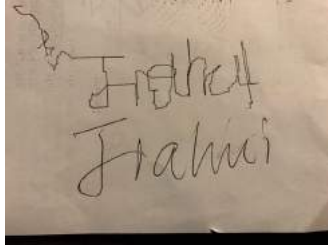


Figure 57: Third Version of Signature



Figure 58: Fourth Version of Signature

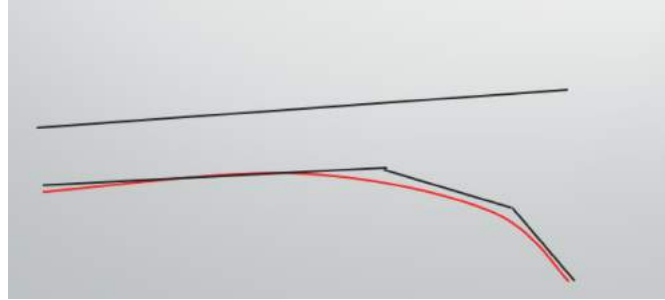


Figure 59: Analysis of Writing a Curve with Speed-Position Algorithm

4.3.3 Bresenham's line algorithm

Methods In order to better write curves, we implemented Bresenham's line algorithm. To calculate the coordinates for a line between two coordinates, the initial (x_1, y_1) and final point (x_2, y_2) is defined. Then, the incremental differences between x_1 and x_2 , as well as y_1 and y_2 can be calculated. This means that to get to x_2 from x_1 , increments of 0.2 is executed. The same applies to the difference between y_1 and y_2 . An increment of 0.2 was determined Leap Motion's distance readings are accurate to 0.2 mm. The error position in x and the error position in y can then be calculated as the difference between the differential increment of x and the differential increment of y , respectively.

To update the coordinate pairs leading to the final point, the algorithm loop starts by determining whether or not point (x_1, y_1) lie at the same location as (x_2, y_2) . If they are two distinguished points, the error position between the two points decreases as follows. If the error position is less than the incremental difference in x , move right. Otherwise, stay in place. If the error position is less than the incremental difference in y , move up. Otherwise, stay in place. Keep repeating this process until the final point (x_2, y_2) and the previous incremental difference point is 0.2. The loop is exited when the final point reads (x_2, y_2) . This coordinate update is coded as shown in Figure 60.

```

// Main loop
while (!((x1 == x2) && (y1 == y2))) {
    var e2 = err << 1;
    if (e2 > -dy) {
        err -= dy;
        x1 += sx;
    }
    if (e2 < dx) {
        err += dx;
        y1 += sy;
    }
    // Set coordinates
    coordinatesArray.push(x1);
    coordinatesArray.push(y1);
}

```

Figure 60: The Main Loop of Bresenham's Algorithm

Results After implementing this algorithm, we can get the results in Figure 61, and our signature looks really smooth when compared to the results of former algorithms. However, when we use this algorithm, we need to write our signature slowly to wait for the system to respond, because it needs some time to iteratively calculate the error position. When we move fast, the result will be unpredictable.

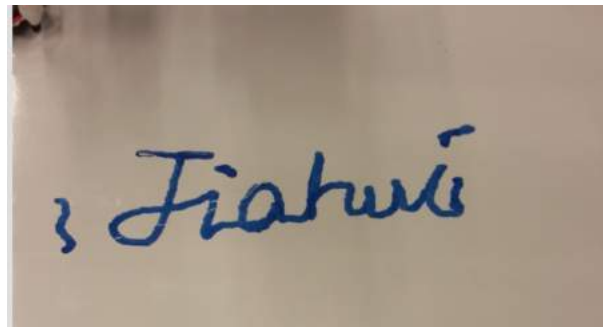


Figure 61: Bresenham's Line Algorithm Result

4.3.4 Modified Bresenham's line algorithm

Methods In order to implement this algorithm, we need to determine different speeds and moving steps of the XY Plotter based on hand speed and error position between two frames. For example, suppose the error position between two frames from Leap Motion is 1 mm in the x direction and 0.5 mm in the z direction. Also the speed along the x axis is 200 mm/s and the speed along the z axis is 100 mm/s. Based on the Results section from Speed-Position Control Algorithm testing, the number of steps along the X direction should be 5 (which is $5 \times 1 = 5$) and the number of steps along the Z direction should be 2.5 (which is $5 \times 0.5 = 0.25$). This movement should be located in Quadrant 1 of a x-y Cartesian coordinate system.

In order to find the right direction to go, the tangent of angle(V_z/V_x) (defined as θ_z) is calculated. If θ_z is larger than 1, the speed of Z will be set as 100 rpm and the speed of X will be set as $100 \times (\theta_z)$ rpm and vice-versa. Then we round the steps to the nearest integer value. Therefore stepper motor X will move 5 steps in the X direction with speed 100 and stepper motor Z will move 3 steps in the Z direction with the speed $100 \times \theta_z$ at the same frame (shown in Figure 62). Table 12 shows the idea of this algorithm.

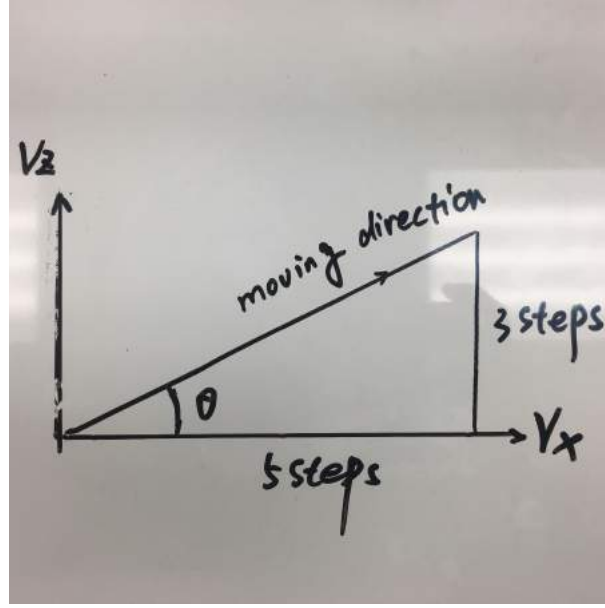


Figure 62: Modified Bresenham's Algorithm

	Speed of X	Speed of Z	Steps of X	Steps of Z
Speed Ratio > 1	$100/\theta_z$	100	$\alpha * errX$	$\alpha * errZ$
$0 < \text{Speed Ratio} < 1$	100	$100 * \theta_z$	$\alpha * errX$	$\alpha * errZ$
$-1 < \text{Speed Ratio} < 0$	100	$100 * \text{abs}(\theta_z)$	$\alpha * errX$	$\alpha * errZ$
Speed Ratio < -1	$100/\text{abs}(\theta_z)$	100	$\alpha * errX$	$\alpha * errZ$

Table 12: Modified Bresenham's Algorithm

Results Figure 63 and 64 are the results obtained from this algorithm. We can write a curve better than other algorithms used before. We test all the letters from a to z. In Figure 65, the black letters are what the XY Plotter writes and the blue letters are what a human wrote. This shows that the writing results of the XY Plotter writes closely follows what a human can write.

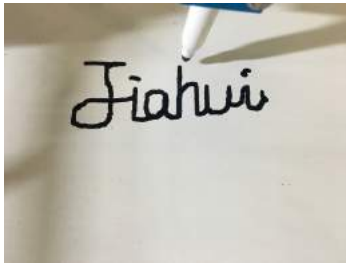


Figure 63: Modified Algorithm Result 1

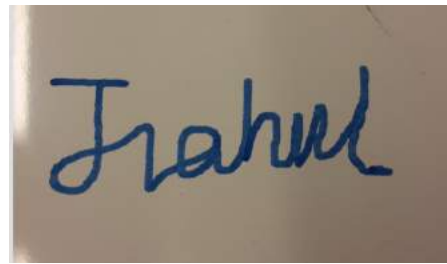


Figure 64: Modified Algorithm Result 2

Although all algorithms are capable of tracking our hand movement and writing our signature, the results of the first two algorithms are not as good as the last two algorithms. The last two results are based on Bresenham's line algorithm; which is why they are similar. Through testing it was determined that the original Bresenham algorithm was computationally heavy, because it requires many calculations within one frame interval. It would consume significant memory and prevent proper operation for fast moving hands. The Modified Bresenham Algorithm has almost the same results and significantly reduces memory consumption. That is why the Modified Bresenham Algorithm is used in our final implementation.

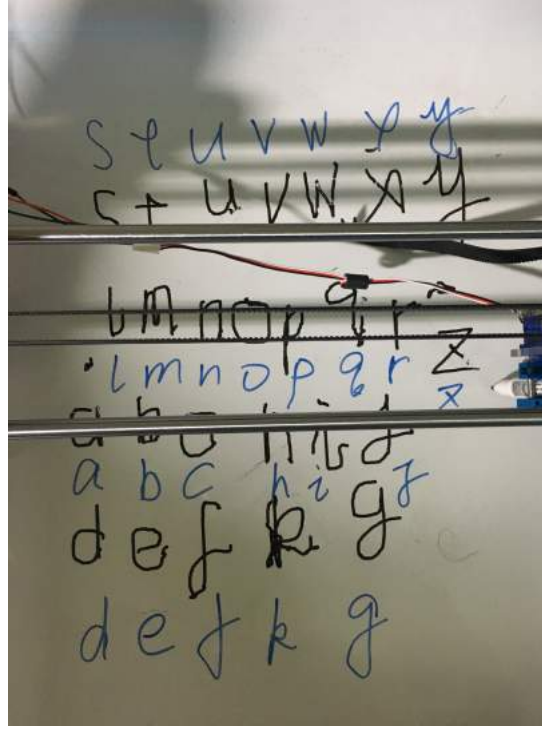


Figure 65: Modified Algorithm Result of Different Letters

Other Language Attempts In order to test whether our algorithm can be implemented in other languages, we also used our system to write some Chinese words. Figures 66 and 67 show fairly accurate results (The upper character in these two pictures is writing result by XY Plotter, and the lower is normal writing result by hand). The result is expected because our system is tracking the movement of human hands. As long as the user can write that language, our system should be able to replicate writing in that language.

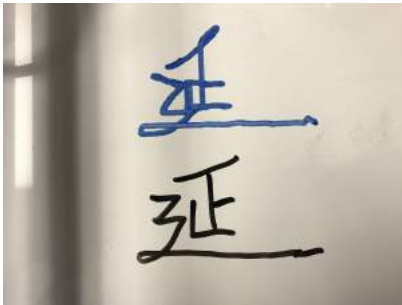


Figure 66: Chinese "Yan" Writing Result

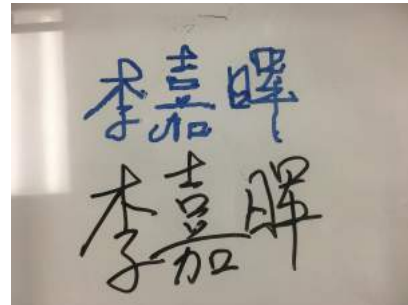


Figure 67: Chinese "LiJiaHui" Writing Result

5 Replication Process

5.1 Robotic Arms Replication Algorithm

5.1.1 Background

Replicating hand writing motion is an integral part of our problem statement. The analytic process of developing a replication algorithm began during the robotic arm development stage of this project. After serious consideration and analysis, using arrays to store motion capture data in JavaScript was hypothesized to best accomplish replication; since it is a useful way to store data in computer languages. An efficient and accurate algorithm was needed to remember and replicate writing gestures through this array utilization.

5.1.2 Methods

Starting with our experimentation with DC motor robotic arms, we used Leap Motion to read hand positions that include x, y, and z axis coordinates. This information is recorded into an array, because the DC motor robotic arm has almost the same DOF as that of the human arm. The Figure 68 shows the general algorithm of recording finger position data implemented in JavaScript.

Then based on our conception, after recording hand motion data, we could begin reading the position data in the array, thus starting the replication process. We were confident that we could implement the replication function as long as the writing process could be accomplished on the DC motor robotic arms. Even though this algorithm was developed for the DC motor robotic arm, it has the potential of being to the XY plotter.

```
// recording index position data
if ((Math.round(hand.fingers[1].tipPosition[0]) % 3 == 0) ||
    (Math.round(hand.fingers[1].tipPosition[1]) % 3 == 0) ||
    (Math.round(hand.fingers[1].tipPosition[2]) % 3 == 0)) {
  index[i] = [Math.round(hand.fingers[1].tipPosition[0]),
              Math.round(hand.fingers[1].tipPosition[1]),
              Math.round(hand.fingers[1].tipPosition[2])];
  console.log('The X, Y, Z of Index: ' + index[i][0] + ',' + index[i][1] + ',' + index[i][2]);
  if (hand.fingers[1].tipPosition[1] < 0){
    console.log('Has negative value');
  } else {
    console.log("did not see negative value");
  }
} else {
  console.log('Skip and no new index data written in the palm array');
}
```

Figure 68: Picture of Algorithm of Finger Position Data Storage

5.1.3 Testing and Results

Due to the various challenges of the OWI robotic arm, including inverse kinematics, writing with the DC motor robotic arm was not achievable. Therefore, this replication process could not be developed further. However, storing motion data in arrays could record hand motion. This shows the feasibility of using arrays to store the motion-related data and use them in the replication process.

5.2 XY Plotter Replication Algorithm

5.2.1 Background

After changing to the XY Plotter system, the types of data that are stored in the arrays needed to change from position data to motor steps and servo motor states. We changed the type of data stored because of three reasons. Firstly it is easy to set the states of the servo motor that control the pen including the pen-up and pen-down states. Secondly, storing stepper motor steps is simpler than storing position data and eliminates the errors from being during the replication process. Thirdly, within one time interval, if position data is recorded the stepper motors need time to move a certain number of steps. However, the position data is recorded in real-time. Thus if the finger tip positions change too fast, the stepper motors may not have enough time to finish their steps. This would make the writing trajectory made by the XY Plotter different from what the human's hand motion is. This causes errors in the writing and replicating process. Therefore, we could not directly use the position data read by Leap Motion to process gesture replication. Based on these considerations and analysis, we decided to record the fixed data of motor steps and servo motor states instead of position data.

5.2.2 Methods

XY Plotter Coordinate System Settings As mentioned in the writing process, we set the coordinate system of the XY Plotter to be the same as Leap Motion's.

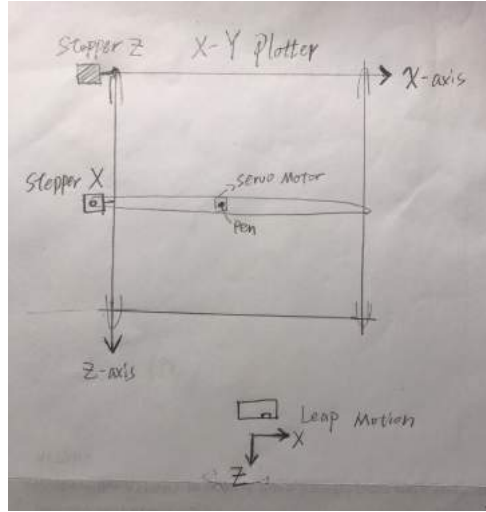
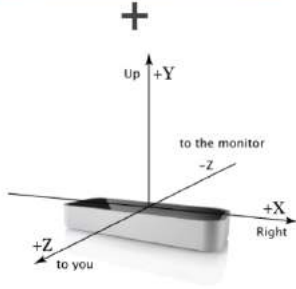
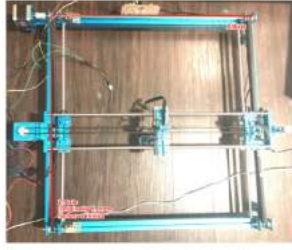


Figure 69: Picture of Combination of XY Plotter and Leap Motion[46]

Figure 70: Top View of XY Plotter Coordinate System

By combining the coordinate systems shown in Figure 69, a new coordinate system for the XY Plotter is developed, as shown in Figure 70. Also in this figure, the servo motor is the only motor that would need to be considered to control the y axis (vertical direction that is pointing out of the screen). This controls the pen-up and pen-down states.

Algorithm of Storing Data into Arrays After reorganizing the system, the algorithm of storing data into arrays could be developed easily. The array storage process coincides with the beginning of the writing process. As mentioned above, we use the leap.js package to obtain index velocity and position data in three axes detected by Leap Motion. This data would set the condition used in our code to begin writing and storing data into an array. We used johnny-five.js to control the Arduino board and motors, and record motors data. Thus, we successfully let the system's writing process history to be recorded.

This process overview demands that a method for storing movement data into arrays be developed. As mentioned, we have three motors on the XY Plotter: two stepper motors to move x and z direction and one servo motor to control the pen up and pen down states in the y direction. Therefore, we need three arrays to record the movement of motors.

a. Start Condition of Storing Array By calling leap.js, we have the opportunities to use hand gestures to trigger the starting condition of recording the data into arrays.

i. Single-hand Start Method: Firstly we used the condition that once a hand is detected (`frame.hands.length > 0`), we begin to start recording the data into arrays.

ii. Two-hand Start Method: Then we decided to utilize the abundant hand gestures provided by leap.js and decided to use a two-hand gesture. That is once you are ready to write/draw and Leap Motion detects there are two hands above it, we could begin to record data into arrays. Thus we can control recording arrays, shrink the replication time, and improve the accuracy and stability of replication analysis.

b. Storing Array of Pen Servo Motor For the pen servo motor, it only needs to record two states into the array: pen-up state and pen-down state.

i. Speed-Position Pen Control Algorithm: As seen in Table 13 below, the conditions for recording the states of the servo motor are listed. When the index tip position is higher than 350 mm on the y axis and its velocity is larger than + 850 mm/s on the positive y axis, we push +1 into the array. This indicates the pen-up state. In this state the servo motor rotates to an angle of 15 degrees. When the index tip position is lower than 280 mm on the y-axis and its velocity is larger than 850 mm/s on the negative y axis, we push -1 into the array. This indicates the pen-down state. In this state the servo motor rotates to an angle of 50 degrees. We push 0 into the array; indicating to keep the previous status.

V = index tip vel P = index tip pos	V > 850 mm/s (on -y axis)	V > 850 mm/s (on +y axis)	Other Condition
0 mm < P < 280mm	-1	0	0
P >= 380mm	0	1	0
Other Condition	0	0	0

Table 13: Method 1 Table of Data Storage in Array of Pen Servo Motor

ii. Tip Position Distance Pen Control Algorithm: In our system, we finally decided to use this method to control the pen-up and pen-down states. That is when the distance of the thumb and middle finger on your writing hand is less than 60 mm, we let the pen go down and push -1 into the array. When the distance is larger than 70 mm, we let the pen go up and push +1 into the array. As previously stated, we push 0 if the status of the pen does not change. The reason for the gap between the 60 and 70 mm threshold is that we want to provide the hysteresis [47] between the two states for more pen control accuracy and stability. This is shown in Table 14.

d: Distance between thumb and middle fingers	d < 60 mm	d > 70 mm	Other Condition
Data in Array	-1	1	0

Table 14: Method 2 Table of Data Storage in Array of Pen Servo Motor

c. Storing Array of Stepper Motors For the stepper motors, its movement is more complicated and sensitive. In the development of this system, we tried three methods (as described in writing process) of storing data into the arrays to replicate hand writing. This is done in order to find better writing and replication results.

i. Speed-Only Control Algorithm: First we combine the codes for writing and replicating. Initially, we only recorded the hand velocity data into the array for replication. However, we found that velocity cannot perfectly control the movement of the pen on the XY Plotter during replication.

ii. Speed-Position Control Algorithm: We then changed from recording the movement of steps to recording every step movement. This accounts for position errors. Then the fixed position data (called total error position data) is stored into the arrays. This more accurately represents our hand movements during writing process. However as mentioned above, the result of drawing a curve is not smooth enough.

iii. Bresenham's Line Algorithm: After researching, we finally decided to use this method in our final implementation. We create two more arrays to record the speed of Stepper motor X and Stepper motor Z in addition to the arrays for total error position of Stepper motor X and Stepper motor Z. We used the recorded speed data in the array to control the movement speed of stepper motors while replicating

and storing the total error position data. This will process the steps that the motors should move during replication.

d. Stop Condition of Storing Array After storing all the data that we need, we need to find a way to stop Leap Motion from recording the data into arrays. We decided to use the command `controllerLM.disconnect()` to shut down Leap Motion. This finishes the array storage and writing process, and the replication process can begin.

To trigger the disconnection of Leap Motion:

i. **Speed Control:** At first we use the condition that a hand palm velocity on the positive x axis larger than 1000 m/s is detected by Leap Motion, the controller disconnects.

ii. **Palm Gesture Control:** We decided to use the gesture provided by Leap Motion called `hand.palmNormal`. Using this gesture, if the palm is facing up, then its normal vector is positive. Once this is detected by Leap Motion, it will shut down and recording data into the arrays stops.

Algorithm of Reading Data from Arrays The significant processes for gesture replication will start right after the Leap Motion controller disconnects.

a. Reset part We realized we need to find a place on the whiteboard for the pen to reset its location after completing the writing process and then start the hand motion copying. The step motor data array is read inversely and the motors rotate in reverse in comparison to the writing process (e.g. writing clockwise (cw), reset counterclockwise (ccw)). In accomplishing this process, we also use the `setInterval()` function in JavaScript to provide the loop function that reads the data in the arrays one by one from the end of datum to the first datum. By setting an array position variable `i` which does `i--` in each iteration until `i = 1`. This provides the time delay that ensures enough processing time for each datum of movement steps made by the motors. Thus, we could implement the result of keeping the pen-up state and following the inverse trajectory of the pen's writing. This returns the pen to the starting point of writing in each replication run.

b. Handwriting Copy Part When `i = 1`, we begin the process of copying/replicating after reset. We add another `setInterval()` function inside of that reset part so we could sequentially read the stepper motor steps and servo motor states from the first datum in the arrays to the end by `i++` in each iteration until `i` equals to `length of arrays - 1`. The steps and angles the motors move correlate to the data read from these motor arrays. Figure 71 below shows the brief algorithm structure of reset and copy parts in JavaScript.

```
var i = 0; // array position
var refreshId1 = setInterval(function() { // 1st setInterval reset
  //Reset part codes here!
  if (i == 1) {
    console.log("end reset");
    j = i;
    clearInterval(refreshId1); // kill the reset
    if (j == 1) {
      var refreshId = setInterval(function() { // 2nd setInterval replicate
        // Copy part codes here!
        i++;
      }, 200);
    }
  }
  i--;
}, 100);
```

Figure 71: Picture of Algorithm Structure of Reset and Copy Parts

c. Start and Stop Replication Part We create an interface based on using the `readline` module in `johnny-five` so that the computer prompts the user to select whether they want to do one (or more) or zero replications each time before the replication process (Figure 72). Thus including the function of

copying the same hand motion multiple times as proposed in the problem statement. The user can stop the entire process after each replication run.

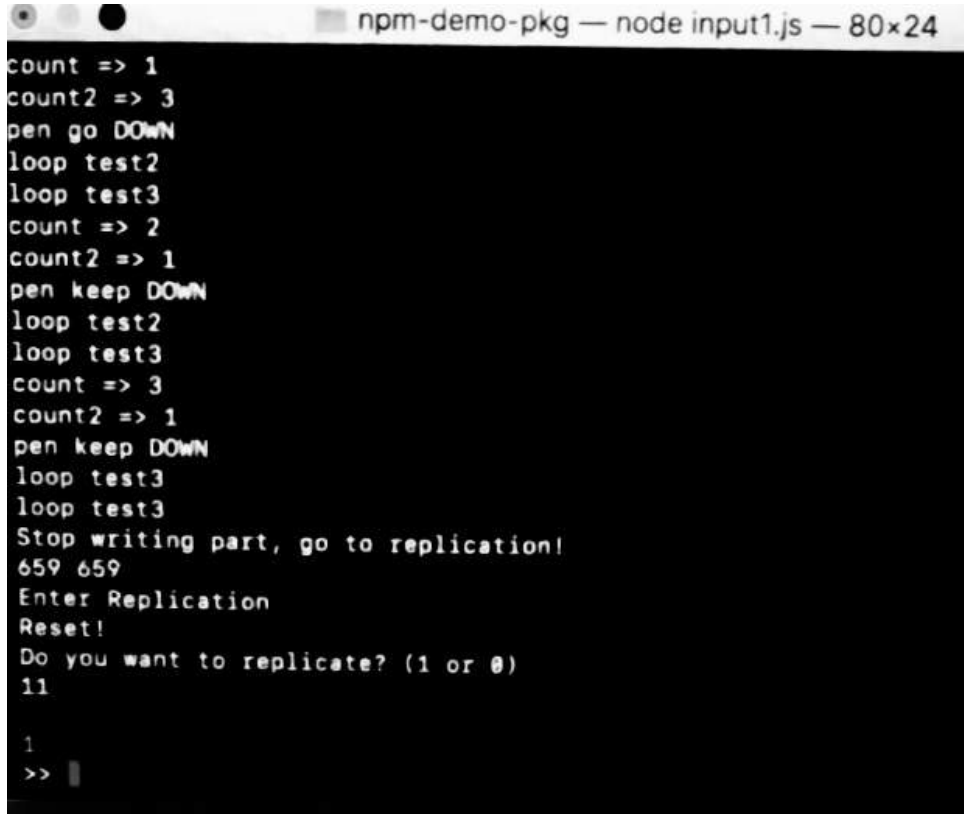
A terminal window titled 'npm-demo-pkg — node input1.js — 80x24' displays a JavaScript script. The script initializes 'count' to 1 and 'count2' to 3, then enters a loop where 'pen go DOWN', 'loop test2', and 'loop test3' are executed. It then increments 'count' to 2 and 'count2' to 1, repeating the loop. This process continues until 'count' reaches 3 and 'count2' reaches 1. After the loops, it prints 'Stop writing part, go to replication!', followed by '659 659'. It then prompts 'Enter Replication' and 'Reset!'. A question 'Do you want to replicate? (1 or 0)' is asked, with '11' entered as a response. The prompt '1' is shown at the bottom, followed by the shell prompt '>>'.

Figure 72: Picture of Interface in Terminal

5.2.3 Testing

Testing of Storing Data into Arrays

Testing Methods The testing of storing data into arrays coincides with the writing process. Specifically for storing array, we begin our writing process in the XY Plotter System to test the Array Storage process. This testing included using different hand gestures to start recording the data into arrays, control the pen-up and pen-down function, and shut down Leap Motion. Also we write some `console.log()` commands in the code to check whether the motion data stored in the arrays are correct.

Results The testing results below shows the process of how we finally optimized our system to current satisfactory status.

a. Result of Testing Hand Gesture to Start Array Storage: In the tests, we found if we use Single-hand Start Method, there is a lot of unnecessary data recorded into the arrays. Firstly, it includes data that is recorded when the hand is only detected by Leap Motion and does not have effective hand-writing motion. Secondly, when finding the position to write and the pen is still moving in the air (pen-up state). Thirdly, any irrelevant hand motion with that is not desired to be recorded into the array. This extends the total replication process time and unnecessary writing motion during replication.

Then we decided to use the Two-hand Start Method which helps to remove the unnecessary data, shrink the processing time in replication, and improve the accuracy of replication. As claimed in the method section, with the `console.log()` command, we obtain the output of the pen servo and stepper motor arrays to the terminal, and we found that the result of storing arrays is optimized compared with Single-hand Start Method. As you can see from the example terminal screen-shots (Figures 73 and 74)

below, unnecessary data is removed (in this case, removing "0" at the beginning of the array).

[illegible]

Figure 73: Example of Pen Servo Motor Single-hand Start Method Result

```
seaseandemacbook-Pro:npm-demo-pkg seaseand$  
seaseandMacBook-Pro:npm-demo-pkg seaseand$ node arraytest.js  
PenServo Data1: 0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,-1,0,0,  
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,  
0,0,0,0
```

Figure 74: Example of Pen Servo Motor Two-hand Start Method Result

b. Result of Testing the Data in the Arrays of Pen Servo Motor: As mentioned above, at first we used the Speed-Position Pen Control Algorithm to set the condition to store the motor state data into an array. However in the test, we found that using the speed and position to be the trigger condition to control the array storage is too sensitive. During the writing process, the pen may accidentally go up and down; causing a serious accuracy problem in the writing and replication process. Therefore we used the tip position distance between thumb and middle fingers to control the pen-up and pen-down states. We found it is more stable and easier to be controlled by the user. Therefore, we finally decided to use this method to control the pen in our system.

c. Result of Testing the Data in the Arrays of Stepper Motors: As mentioned in the section of Storing Array of Stepper Motors, our testing of recording data from stepper motors included three different methods. These three methods gradually changed during our testing process because of optimization to our writing process. Coinciding with testing of the Bresenham Algorithm for writing, we checked the data in the stepper motor and motor speed arrays by outputting them in the terminal (Figures 75 and 76) and the results are reasonable. Also the result of writing and replication motion trajectory shown on the screen and whiteboard are more satisfactory. We will discuss more about the writing and replicating results in the testing of reading array part.



Figure 75: Example
Steps of Stepper X

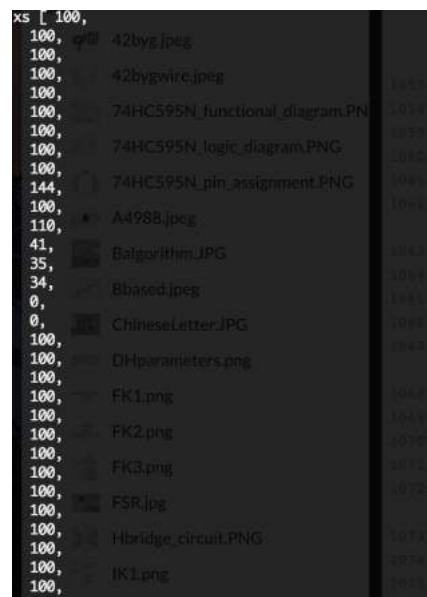


Figure 76: Example of Array of Motor Speed of Stepper X

d. Result of Testing Hand Gesture to Stop Array Storage: In the tests, we found if we use the condition of 1000 mm/s horizontal palm velocity to stop Leap Motion, the unnecessary motion data would also be recorded into the array while swiping our hand to reach that speed. This causes longer processing time because we have to read longer arrays and the inaccuracy and instability of replication because of a quick swiping hand. Thus we changed to detect the palm states, and as we mentioned in the part of Algorithm of Storing Data into Arrays, only when changing the state of the palm from facing down to facing up, Leap Motion will disconnect and the arrays stop recording data. We found it is a really efficient gesture that could stop the arrays quickly and is not mistakenly triggered because of sensitivity. Therefore, we finally used this gesture as our optimal method to turn off the writing process and Array Storage.

Overall, with optimized methods for the start and stop trigger conditions, the issues of unnecessary data storage and inaccuracy of motion recording are solved completely. Our system operation then became more stable and reasonable.

Testing of Reading Data from Arrays

Testing Methods The testing methods here will cover the methods regarding Reset Part, Handwriting Copy Part, and Start and Stop Replication Part.

a. The method we are using to test for the Reset Part is comparing the locations of the writing and copying starting points. If the two locations are relatively close, we will announce that the process of Reset Part is successful. When the condition of moving the pen to a suitable place on whiteboard is achieved, the system is ready to start copying the writing results.

b. The methods we mainly used to test the process titled Handwriting Copy Part in Reading Data from Arrays are comparing the similarity of writing and copying results. To examine similarity, first we compare the size of the results between the writing process and copying process. Second we compare the similarity of writing and replicating trajectory.

As for the way we calculated the size of the results, we pick the results' longest vertical distance from top to bottom to be their width, and pick their longest horizontal distance from left to right to be their length. Then we calculate the area of the square that includes the entire writing result (Figure 77).

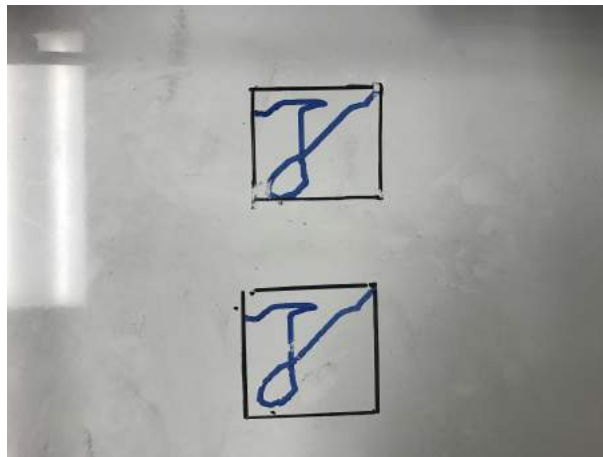


Figure 77: The Comparison between Writing and Replicating Results Size

In terms of comparing the similarity between the writing and replicating trajectory, as shown in Figure 78 below, we can compare the two results by overlapping the two trajectories together. This makes it easier to directly compare the writing and replicating results.

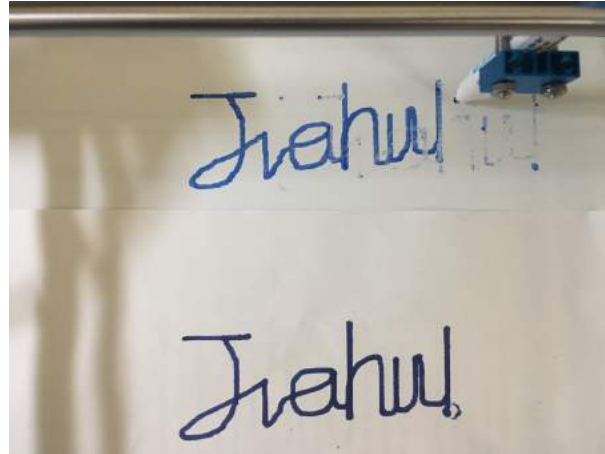


Figure 78: The Example Used in Comparing Trajectories of Writing and Replicating Results

c. The method used for testing the Start and Stop Replication Part, we input Yes ("1"), No("0"), and other characters to see whether the computer will correctly process our commands for doing replication process or not.

Results The Results includes Reset Part, Handwriting Copy Part, and Start and Stop Replication Part of Algorithm of Reading Data from Arrays.

a. The Result of Testing Reset Part:

During the replication process, our Reset Part is able to return to the starting location of the writing process; meaning that it does not need much more optimization. As you can see in Figure 79 our goal is almost accomplished. One thing that is worth mentioning is that in the following test results we moved the copy part away from the location that the writing process begins. This is because we want to show more clear and comparable results in the report. In demonstration or further testing, we would provide replaceable thin whiteboards under the pen so that we could replace from one to another after each writing iteration and before each replication run.

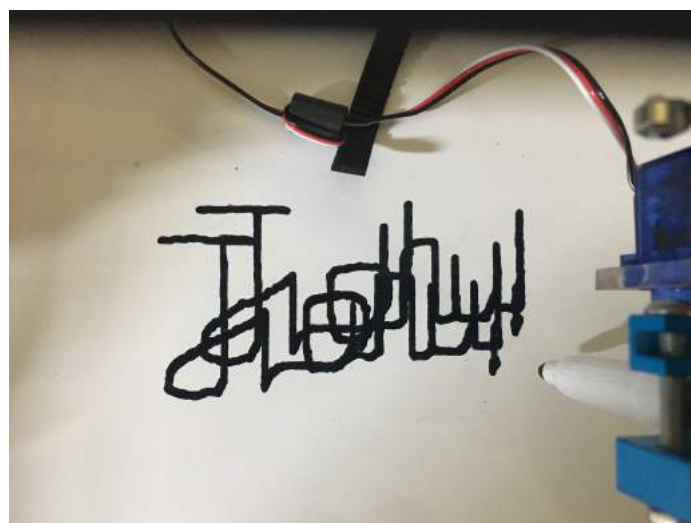


Figure 79: The Picture of Reset Part Test

b. The Result of Testing Handwriting Copy Part:

From the tests of Handwriting Copy Part, we found the size and trajectory of the copy result is always smaller than the original writing part. For Figure 81, the writing result is the upper one, and the copy result is the lower one. From the picture, you can see the horizontal line at the top of "J" and the loop at the bottom left of the copy result are all smaller compared to the writing result. Then we calculate the size ratio between the area of the copy part and the writing part:

$$Area_{Copy}/Area_{Writing} = (22mm * 40mm) / (23mm * 42mm) = 0.91$$

Figure 80: Not-Optimized Size Ratio Calculation Result of Figure 81

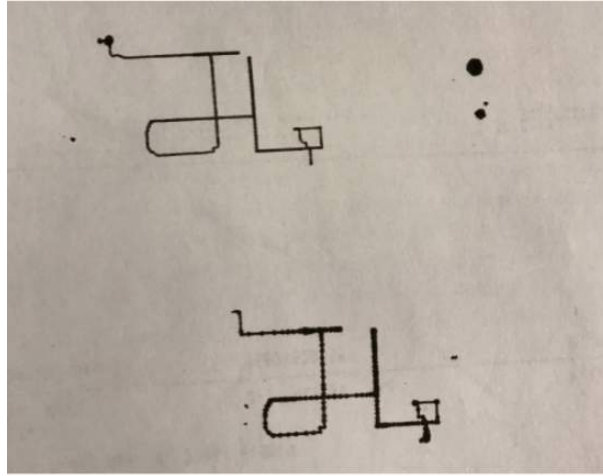


Figure 81: Picture of Writing and Copying of "JHL"

From the calculation results, we find that the size ratio is 0.91. Although the difference between the two results is small, the size and trajectory differences are not negligible. We believe this issue of dissimilarity could be solved and the results could be optimized.

We then analyzed the results and determined the reason for this problem is friction between the paper and the real base (e.g. desk, table, etc) and between the pen and paper. In the copy part, the speed of the motor is relatively lower than that of the writing part. This is to make the replication process more stable and the result more accurate. However this gives rise to the relative movement between the real base and paper because the friction between the pen and paper becomes static friction; as opposed to kinetic friction in the writing part (static friction > kinetic friction). [48]

As you can see in Figure 81, there are many staggered points in the copy part (the lower result), and all of them are causing the transient condition of static friction. Therefore, the pen and paper move together in those time intervals and cause relative motion between the paper and real base. If these static friction errors accumulate, the length of motion trajectory and size of the copy result are smaller. If the pen contacts the paper even harder, this condition becomes worse.

Also in the processes of testing, we had excessive paper waste because of performing many writing and replicating test trials (Figure 82). This complicates the group's testing processes. This could also be a serious waste issue for others who may use our design in the future.



Figure 82: Picture of Paper Waste

For solving friction and paper waste issues, we decided to use a whiteboard instead of paper. By using a wooden base under the whiteboard or a large surface whiteboard, we can decrease the friction coefficient and the relative movement between the whiteboard and real base. Also we decrease the contact area between the pen and whiteboard. This is done by tuning the relative position of the pen and servo motor to decrease the normal pressure. Thus decreasing the friction and getting rid of the possibility of static friction occurring between the contact points. Therefore, the size difference problem is solved. Using an erasable whiteboard is also more environmentally conscious for our system design to save more paper in research, testing, and further development.

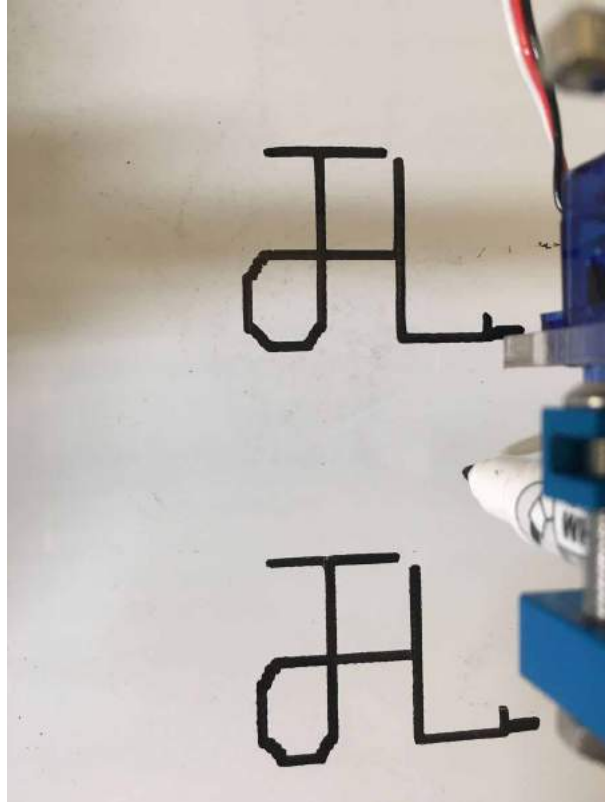


Figure 83: Picture of Optimized Writing and Copying result of "JHL"

After optimizing, we calculated the size ratios of the writing and testing results again and compared the similarity of the trajectories of the two results. To evaluate our optimization, we calculate the size ratios of the results shown in Figure 77 and Figure 83. In addition, Figure 86 overlaps the writing and replicating results of Figure 78. This further depicts our optimization because the trajectories of the two results overlap each other very closely.

$$Area_{Copy}/Area_{Writing} = (24mm * 42mm) / (24mm * 43mm) = 0.977$$

Figure 84: Optimized Size Ratio Calculation of Figure 83

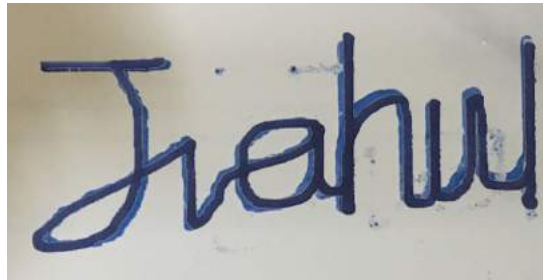


Figure 86: Picture of Comparison of Trajectories of Optimized Writing and Copying results

c. The result of Start and Stop Replication Part:



Figure 87: Picture of Testing Result of Interface in Terminal

As you can see in the Figure 87 shown above, we successfully accomplished the function of letting the computer prompt the user whether he/she wants to do (multiple) replication or no replication. Therefore, the user is able to let the system stop working after writing process or before each run of replication process.

6 Conclusions

6.1 Summary of Key Design Changes

For the software implementation, we have changed our robotics framework from Cylon to johnny-five. For the system implementation, we have changed our actuators from servo motor robotic arm to the OWI robotic arm to the XY Plotter. For the XY plotter algorithms, we have changed from the speed-only control algorithm to speed-position control algorithm to Bresenham's line algorithm to the modified Bresenham line algorithm.

6.2 Key Learned Concepts

Throughout the development of this project, we have been exposed to 3D sensing through the use of cameras and thorough hand data analysis. We have also studied actuators response to 3D technology (i.e. indirect physical stimuli). Through testing, we've analyzed the accuracy and limitations of different motors to move a system. We have become more adept in coding in JavaScript and using the runtime environment Node.js to execute our software implementation. Additionally, we have developed a more thorough understanding of the capabilities of inverse kinematics. During our work with the OWI arm, different methods of inverse kinematics (including geometric approaches) were analyzed. We were introduced to the Bresenham's line algorithm for linear approximation of curves.

6.3 Performance Analysis

Hand writing is unique to each individual because it is a consequence of cognitive as well as physical characteristics that vary from person to person. Our group utilized robotics, motion capture devices, and software implementation to accurately reproduce hand writing gestures.

By transferring the robotic system from the OWI arm to the XY Plotter, our system was able to increase its accuracy to translate hand detection to motor rotation with thorough data synthesis. This was verified with various testing experiments that include physical dimension examinations and analyzing the legibility of words and drawings made with the XY Plotter.

Our system was able to accurately replicate hand movements because the development of the replication process follows the development of the writing process. The legibility of the XY Plotter's replication is affirmed through our ability to distinguish individual letters. During the analysis of Bresenham's Line Algorithm, we were able to devise with a computationally efficient algorithm that uses less memory while still capable of accurate functionality.

6.4 Broader Implication

The development of this project falls under the category of motion capture, 3D, and virtual reality technology; an evolving technology and market sector that integrates significant user interaction. This project demonstrates an integration of motion capture technology in a method that has not been widely experimented with. This means that other members of the scientific community could use this project as a basis for more accurate development with access to more advanced technology and analysis tools. In the future, using touchpad technology instead of physical writing surfaces would store results of handwriting electronically and reduce waste.

All components in our systems are market-available products. The XY-plotter implementation can be attained with less than a \$400 investment. Much of the software implementation had online resources

in the form of packages, libraries, and open-source community information to assist with project development. This means that this is a feasible implementation and project for most of the consumer and academic population because of its easy accessible resources.

7 Supplementary Materials

7.1 Github Repository Link

This is a link to our Github page containing all the source code we have used in our project: [EE209 Project Repository](#)

7.2 Demonstration Video

This is a link to a demonstration video that shows the instructions on how to use the system, as well as demonstrating the functionality of our project: [EE209 Project Demo](#)

7.3 User Interface

This is a link to our User Interface Page: [EE209 Project User Simulation Interface](#)

References

- [1] Artal-Sevil, J.S., and J.L Montanes. "Development of a Robotic Arm and Implementation of a Control Strategy for Gesture Recognition through Leap Motion Device." IEEE Xplore Digital Library. IEEE Xplore, 22 June 2016.
- [2] "Leap Motion CHOP," Leap Motion CHOP - TouchDesigner 088 Wiki. [Online]. Available: https://www.derivative.ca/wiki088/index.php?title=Leap_Motion_CHOP.
- [3] "Getting Started," Getting Started Javascript | Leap Motion Developers. [Online]. Available: <https://developer.leapmotion.com/getting-started/javascript>.
- [4] "ECMA-404 The JSON Data Interchange Standard," *Introducing JSON*. [Online]. Available: <http://www.json.org/>. [Accessed: 07-Nov-2016].
- [5] ArduinoOrg. "Arduino UNO." Arduino - Open Source Products for Electronic Projects.[online]. Available: <http://www.arduino.org/products/boards/arduino-uno>.
- [6] "Arduino - ArduinoBoardUno." Arduino - ArduinoBoardUno.[online]. Available: <https://www.arduino.cc/en/Main/ArduinoBoardUno>.
- [7] N. Foundation, "Node.js," Node.js. [Online]. Available: <https://nodejs.org/en/>.
- [8] "jQuery.ajax()," *jQuery API Documentation*. [Online]. Available: <http://api.jquery.com/jquery.ajax/>. [Accessed: 07-Nov-2016].
- [9] "syntax error: unexpected end of input parseJSON," Stack Overflow Community, 14-Mar-2015. [Online]. Available: <http://stackoverflow.com/questions/25173727/syntax-error-unexpected-end-of-input-parsejson>. [Accessed: 08-Nov-2016].
- [10] "Frame," *Frame — Leap Motion JavaScript SDK v3.1 Documentation*, 2012. [Online]. Available: <https://developer.leapmotion.com/documentation/javascript/api/Leap.Frame.html>. [Accessed: 29-Oct-2016].
- [11] "Hand," *Hand — Leap Motion JavaScript SDK v3.1 Documentation*, 2012. [Online]. Available: <https://developer.leapmotion.com/documentation/javascript/api/Leap.Frame.html>. [Accessed: 29-Oct-2016].
- [12] R., "rwaldron/johnny-five," GitHub. [Online]. Available: <https://github.com/rwaldron/johnny-five/blob/master/README.md>.
- [13] J. Asia, *Modifications to Robot Arm for Opto Coupler Feedbackc, OWI 535, Edge, etc.* Autodesk.

- [14] *OWI Robotic Arm Motor*. Christmas Toys. [online]. Available: <http://christmastoy-store.com/owi-robotic-arm-edge-product-review/>.
- [15] B. Lawson, "Electric Drives - DC Motors (Description and Applications)," *Electropaedia*, [Online]. Available: <http://www.mpoweruk.com/motorsdc.htm>
- [16] "Motor Shield," *Arduino Playground - AdafruitMotorShield*. [Online]. Available: <http://playground.arduino.cc/Main/AdafruitMotorShield>.
- [17] L. Fried, "Adafruit Motor Shield." Adafruit Learning System, New York City, 06-Jul-2015.
- [18] "Dual H-Bridge Motor Driver for DC or Steppers - 600mA - L293D," *Adafruit Industries Blog RSS*. [Online]. Available: <https://www.adafruit.com/product/807>. [Accessed: 02-Nov-2016].
- [19] A. Tantos, "H-Bridges-The Basics," *Modular Circuits*, 2011. [Online]. Available: <http://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/> [Accessed: 03-Nov-2016].
- [20] "The Half-Bridge Circuit Revealed," *PowerGuru - Power Electronics Information Portal*, 30-Aug-2012. [Online]. Available: <http://www.powerguru.org/the-half-bridge-circuit-revealed/>. [Accessed: 03-Nov-2016].
- [21] "L293x Quadruple Half-H Drivers." Texas Instruments, Dallas, Jan-2016.
- [22] "74HC595 8-bit Serial-In, serial or parallel-out shift register with output latches; 3-state." NXP Semiconductors, Eindhoven, 25-Feb-2016.
- [23] "Shifting Out and the 595 Chip," *Serial to Parallel Shifting-Out with a 74HC595*. [Online]. Available: <https://www.arduino.cc/en/Tutorial/ShiftOut>. [Accessed: 18-Nov-2016].
- [24] "Arduino-AnalogRead," *Arduino Reference*. [Online]. Available: <https://www.arduino.cc/en/Reference/AnalogRead>. [Accessed: 04-Nov-2016].
- [25] J. Hoefs, "Firmata/Protocol," *GitHub*, 15-Aug-2016. [Online]. Available: <https://github.com/firmata/protocol> [Accessed: 30-Oct-2016].
- [26] J. Hoefs, "Firmata/Arduino." *GitHub*, 03-Nov-2016. [Online]. Available: <https://github.com/firmata/arduino> [Accessed: 04-Nov-2016].
- [27] "JavaScript Robotics: Motor API (Johnny-Five)," *Johnny-Five*. [Online]. Available: <http://johnny-five.io/api/motor/>. [Accessed: 29-Oct-2016].
- [28] R. Waldron, "rwaldron/johnny-five/Sensor," *GitHub*, 15-Aug-2016. [Online]. Available: <https://github.com/rwaldron/johnny-five/wiki/Sensor>. [Accessed: 30-Oct-2016].
- [29] A. Knorig and P. Ehrlich, "Unexpected end of JSON input · Issue 219," *leapmotion/leapjs*, 28-Jan-2016. [Online]. Available: <https://github.com/leapmotion/leapjs/issues/219>. [Accessed: 08-Nov-2016].
- [30] S. M., A., and J. Ward, "Sample Period/Frames frequency," Leap Motion Community, 22-Aug-2015. [Online]. Available: <https://community.leapmotion.com/t/sample-period-frames-frequency/3281>. [Accessed: 29-Oct-2016].
- [31] "Controller," Controller — Leap Motion JavaScript SDK v3.1 documentation. [Online]. Available: <https://developer.leapmotion.com/documentation/javascript/api/Leap.Controller.html>. [Accessed: 01-Nov-2016].
- [32] Bruno Siciliano, Lorenzo Sciacivco, Luigi Villani, Giuseppe Oriolo. Robotics modeling, planning and control. (2009)
- [33] A. Khatamian, "Solving Kinematics Problems of a 6-DOF Robot Manipulator," Int'l Conf. Scientific Computing, pp. 228-233, 2015.
- [34] C. Anderson, "Control an OWI Robotic Arm with Arduino," Instructables, 2011. [Online]. Available: <http://www.instructables.com/id/Intro-and-what-youll-need/?ALLSTEPS>. [Accessed: 25-Oct-2016].

- [35] "XY Plotter Robot Kit (with Electronic Version)," Makeblock XY Plotter Robot Kit - DIY plotter - Drawbot Robot Kit STEM . [Online]. Available: <http://www.makeblock.com/xy-plotter-robot-kit>.
- [36] "Makeblock 42BYG Stepper Motor," Studica. [Online]. Available:<http://www.studica.com/us/en/Makeblock/42byg-stepper-motor/>
- [37] "Stepper motor", Diagram of stepper motor. [Online]. Available: [http://www.micropik.com/PDF/42byg\[1\].pdf](http://www.micropik.com/PDF/42byg[1].pdf)
- [38] "AdvancedFirmata" GitHub. [Online]. Available: <https://github.com/firmata/ConfigurableFirmata>
- [39] "rwaldron/johnny-five," GitHub. [Online]. Available: <https://github.com/rwaldron/johnny-five/blob/master/docs/stepper-driver.md>.
- [40] "servo motor" Picture of servo motor [Online]. Available: <http://www.micropik.com/PDF/SG90Servo.pdf>
- [41] "Me Orion (Base on Arduino UNO)," Me Orion (Base on Arduino UNO). [Online]. Available: <http://www.makeblock.com/me-orion-base-on-arduino-uno>.
- [42] "Amazon Glass Ergonomic Tabletop Riser," Amazon.com. [Online]. Available: <https://www.amazon.com/Ergonomic-Tabletop-Computer-Notebook-STAND-V000R/dp/B01A15H2ZE>
- [43] Amazon., "Magnetic Dry Erase White Board with Stain Resistant Technology 17x11" Amazon.com. [Online]. Available: https://www.amazon.com/gp/product/B01HC3EEAE/ref=oh_aui_detailpage_o00_s00?ie=UTF-8
- [44] "index finger motion," CodePen. [Online]. Available: <https://codepen.io/Seasean/pen/GNQdBY/>
- [45] E. Bresenham Algorithm. "Algorithm for computer control of a digital plotter". IBM System journal Vol. 4 No.1 1995
- [46] "Getting Started," Getting Started Javascript | Leap Motion Developers. [Online]. Available: <https://developer.leapmotion.com/getting-started/javascript>.
- [47] "Hysteresis", Hysteresis Wiki. [online]. Available: <https://en.wikipedia.org/wiki/Hysteresis>.
- [48] "Friction", Friction Wiki. [online]. Available: <https://en.wikipedia.org/wiki/Friction>.